SOLIDIFIED

Audit Report for Aventus Protocol. May 11, 2018.

Summary

Audit Report prepared by Solidified for Aventus Protocol Foundation covering their suite of smart contracts which implement a protocol for ticket exchange that eliminates uncontrolled resale and counterfeit tickets.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below contracts. The debrief took place on May 11, 2018 and the final results are presented here.

Audited Files

The following files were covered during the audit:

- AppsManager.sol
- AventusStorage.sol
- AventusVote.sol
- EventsManager.sol
- MultiAccess.sol
- ParameterRegistry.sol
- interfaces/IAppsManager.sol
- interfaces/IAventusStorage.sol
- interfaces/IAVTManager.sol
- interfaces/IERC20.sol
- interfaces/IEventsManager.sol
- interfaces/IProposalsManager.sol
- libraries/LApps.sol
- libraries/LAventusTime.sol
- libraries/LChallengeWinnings.sol
- libraries/LEvents.sol
- libraries/LLock.sol
- libraries/LProposal.sol
- proxies/PApps.sol
- proxies/PAventusTime.sol
- proxies/PDelegate.sol
- proxies/PEvents.sol
- proxies/PLibraryDelegate.sol
- proxies/PLock.sol
- proxies/PProposal.sol

SOLIDIFIED

Audit Report for Aventus Protocol. May 11, 2018.

Notes:
The audit was performed on commit `0285767783e030699e4fb57787e80a17bbae6827`
The audit was based on the solidity compiler `0.4.23+commit.124ca40d`

Intended Behavior

The purpose of the contracts is to implement the Aventus Protocol, as described in the relevant whitepaper.

Issues Found

Critical

No critical vulnerabilities were identified.

Major

1. `distributeWinningsAmongVoters` can run out of gas for a large number of voters

The function `distributeWinningsAmongVoters` loops over all revealed winning voters and adjusts deposits to account for winnings via a call to `giveWinningsToStakeHolder`.

Since the number of winning voters is effectively unbounded, the loop in this function can be considered to be unbounded. If there are too many winning voters, calling this function could no longer be possible within the networks block gas limit.

Recommendation

Consider allowing this function to pay winners in batches, tracking which winners have been paid out in each batch, and thus avoiding double pay-outs. Also consider pull over push, as described in the best practices.

AMMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

2. Library functions are not always upgradeable

The bytecode of Library functions that are marked as internal gets inlined to the bytecode of any contract that references the library, and thus cannot be modified for any upgradeability reasons.

Recommendation

Consider changing any necessary `internal` functions to `public` or `private` in order to make them upgradeable.

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

3. Challenge rewards do not match whitepaper

In the whitepaper, Equation (1) describes the business logic to be followed for distributing the losing party's deposit to the winning party. This mechanism is not implemented in the provided smart contracts.

Recommendation

Update the reward calculation mechanism to match the whitepaper.

Client's Response

The Aventus team has acknowledged the finding and informed the whitepaper will be amended in the upcoming version.

4. Storage data can be modified by owner arbitrarily

All values in `AventusStorage.sol` can be modified by the contract's owner.

Recommendation

Remove `msg.sender == owner` condition from `isAllowedAccess()` in `MultiAccess.sol`.

This ensures that the owner does not have the additional permissions required to arbitrarily modify the contract's storage. In order to maintain upgradeability, consider creating a `AventusUpgradeManager.sol` contract that can modify the contract addresses in storage of libraries.

Client's Response

The Aventus team states that Aventus is a trusted party by design.

5. AVT price is constant

The price of AVT is set once through the `ParameterRegistry` and is never adjusted again.

Recommendation

Use an oracle service whenever the values of AVT in US cents are needed through a service such as Oraclize.it.

Client's Response

The Aventus team acknowledged the finding, and informed that in current versions Aventus will implement a JS script to upload the rates. A trusted oracle is part of the roadmap and will be implemented in future versions.

Minor

6. User cannot vote on multiple proposals

If a user votes on multiple proposals, their full amount of staked AVT will be "used" against their first revealed vote:
`LProposal.sol:L249:uint stake = s.getUInt(keccak256("Lock", "stake", voter));`

This may be unexpected to the user if in practice they are actually participating in multiple votes, and want to split their stake across these multiple votes. It may be better to allow the user to specify the amount they want to stake against a specific reveal through a parameter to `revealVote`.

Recommendation

Consider allowing users to vote on multiple proposals by specifying their stake per vote.

Client's Response

The Aventus team clarified that by design the voters will be able to use the same stake towards all votes casted.

7. `AventusVote` can change storage contract address

`AventusVote` allows the storage address to be modified through `updateStorage`. None of the other instances allow this "in-place" updating of storage (the underlying contracts could be redeployed referencing a different storage location). This is presumably because the `AventusVote` contract references storage directly as well as passing it through to the underlying library contracts (which are actually proxy contracts).

Recommendation

Given that all of the other contracts (e.g. `AppsManager`, `EventsManager`) would need to be redeployed were the storage location to change, consider making the approach with `AventusVote` consistent and remove `updateStorage` and `setStorageOwner`.

8. Transaction Costs can disincentivize voting on challenges

From a game theoretical perspective, the system requires that the community will be voting and challenging depending on what they want to happen. It was found that the total transaction costs for a user's voting process can be more than the amount of funds that they get as a result of the staking. Depending on network conditions (gas price), market conditions (ether price in usd) and challenge conditions (deposit price, total stake involved, percentage of loser's deposit that goes to winner and challenge ender), a user may end up not being compensated sufficiently for voting.
As a result, a user is required to have large stakes locked in the contract so that they have positive expected returns. In addition, the voting process lasts 7 days for the voting period and another 7 days for the reveal period, which exposes users to the volatility of the currency. Finally, if an event gets created with a very large deposit (from an actor such a big institution which can afford it), it is harder for a random actor to challenge that.

Recommendation

There is no straightforward way to solve this other than optimizing the gas costs of the contracts and properly adjusting the reward model to incentivize voting in all cases.

Client's Response

The Aventus team clarified that the gas cost is a known limitation of the system, and that users are expected to bear the gas cost when casting governance votes in the platform.

9. Unable to modify fixedDepositAmount without loss of funds

In LApps.sol, if the value of `fixedDepositAmount` were to change between an App being registered, and de-registered, then it is possible that the `ExpectedDeposits` value is inconsistent between the registering and deregistering of an App.

This could result in an incorrect value being unlocked (through modification of `ExpectedDeposits`).

Recommendation

It may be better to record the actual fixed deposit amount used for a given App registry and unlock this recorded amount rather than relying on the value of `fixedDepositAmount` not changing (whilst changes to this value are controlled by Aventus, so can not be maliciously changed, it may be that Aventus want to modify this value themselves).

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

Notes

10. Use more consistent variable naming

There can be confusion around variable naming. As an example, variables that represent the `AventusStorage` contract are referenced as `s`, `_s`, `_storage`, which can create confusion, especially when combined with the `v`,`r`,`s` values from the ECDSA algorithm.

Recommendation
Follow the same rules for variable naming throughout all the contracts.

11. `Owned` contract implements `kill` function

The implemented version of the `Owned.sol` contract implements `kill` which is unnecessary and potentially dangerous. The function can be called by a potentially malicious owner and delete all the data included in the protocol.

Recommendation
Consider using OpenZeppelin's `Ownable.sol` (which does not include `selfdestruct` functionality) instead of your version.

12. Move all string hash key values to global variables

Currently, accessing a storage variable is done by hashing the key values. There is room for developer error in case there is a typo in the key name, as well as attack surface for a homoglyph attack from a malicious developer. Consider moving all strings to constant variables to the top of the file and using these instead.

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

13. `LAventusTime` is redundant

The `LAventusTime.sol` library contract's implementation is used only for returning the current time, which can be achieved by using `now`. Unless it is going to be replaced with more sophisticated business logic, consider deleting the contract and replacing all occurrences of its usage with `now` (or a mocked version of `now` as needed for test cases).

Client's Response

The Aventus team informed that the library is used for test purposes only, and will be removed from production version.

14. Make setting and getting values from storage more consistent

In contracts such as `libraries/LEvents.sol` a `setEventDeposit` function and a `getEventDeposit` function are implemented, however values for the deposit are accessed manually by calling `_storage.setUInt(keccak256("Event", _eventId, "deposit"), _deposit)` and `_storage.getUInt(keccak256("Event", _eventId, "deposit"))` respectively.

It is recommended to either use the getter/setter functions in all cases to avoid the need for manually writing of the statements, or to remove them from the functions as they increase the bytecode size, and gas costs. This happens in multiple places throughout the contracts.

15. Using constructors without the constructor keyword is deprecated

Using constructors that have the same function name as the contract is deprecated in the latest version of solidity (0.4.23). Consider using the constructor keyword instead.

16. Consider using latest version of solidity

The contracts use solidity version 0.4.19. It is suggested to use the latest version (`0.4.23`) and fix all compiler warnings that arise.

17. Replace wrongly made asserts and add error strings to require statements

Since version `0.4.22` of solidity, `require` statements can include an error string. Consider adding appropriate error messages to the require statement. The two `assert(false)` statements should also be replaced with `revert('Error string')` in LLock.sol, as these are checking user inputs rather than contract invariants.

18. Use `SafeMath` for arithmetic operations

In multiple occasions, arithmetic operations are left unchecked, or the result is manually asserted. Although no overflows or underflows were identified during the audit, it is strongly recommended to replace all arithmetic operations to use `SafeMath`, to ensure all arithmetics invariants are valid. This makes reasoning about the code easier and less error prone.

Client's Response

The Aventus team acknowledged the issue, but informed the use of SafeMath is not planned for the in-scope contracts.

19. Refactor signature checking mechanism

Currently, signature checking for messages is done by extracting the `v`,`r`,`s` parameters of the signed message in the client side, and then verifying the signature parameters in the smart contract through ecrecover. The method is correct, however it requires using helper functions on the client side, along with managing the parameters of the signature. Instead, consider using `ECVerify.sol` in your smart contracts in order to be able to pass a signed message to the smart contract and let it do the verification. This improves UX and reduces offchain client overhead. This can also be used to extract the signing logic to the main contracts and increase code reuse, instead of performing signature validation on the libraries.

20. Consider using `external`

Consider using `external` for function visibility if the method will only be accessed from outside. This can help save some gas.

21. Remove duplicate functions

In `LEvents.sol`, `getExistingEventDeposit` is identical to `getEventDeposit`. In addition, there are two implementations for `getEventDeposit` which have different usage and are not clearly differentiated. Consider removing duplicate function and renaming one of the two identically named functions for clarity.

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

22. Redundant modifier `onlyEventChallengeProposal` in `endEventChallenge`

`endEventchallenge` is a private function that gets called only through `endProposal` if `proposalIsEventchallenge` is true. The modifier `onlyEventChallengeProposal` makes the same check twice and is not required.

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

23. `giveWinningsToStakeHolder` and `giveFixedWinningsToProposalWinner` can be combined

These functions from `LChallengeWinnings`.sol can be merged into one since their functionality. Consider using only `giveWinningsToStakeHolder` and calling it with the appropriate percentage of winnings for distribution. Also consider renaming `stakeLockKey` and `stakeLock` to `depositLockKey` and `depositLock`.

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

24. Use more appropriate data types

In `LLock.sol` `string` variables are used to check if the user wants to deposit or stake his funds. Instead of that, consider using a `boolean` variable `stake`, which will determine that the user wants to stake if `true`, or deposit if `false`. The same applies for voting, instead of using a uint optId which can be 1 or 2, make it a `boolean` variable `agree`, which will determine that the user agrees with a proposal if `true`, or disagrees if `false`. In case there are more than one states, consider using `enum` type variables, instead of strings.

Client's Response

The Aventus team acknowledged the issue. The issue has been partially fixed by the Aventus team by using uint8 for option Id in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

25. Use correct ordering for modifiers and functions

Ideally functions, variables, modifiers and constructors should be declared in a consistent order, following best practices from the documentation.

For example, in `LEvents.sol`, modifiers and functions are interleaved through the code, e.g.:
`LEvents.sol:30 - modifier eventHasDeposit(IAventusStorage _storage, uint _eventId)`
`LEvents.sol:43 - function eventActive(IAventusStorage _storage, uint _eventId)`
`LEvents.sol:59 - modifier inReportingPeriod(IAventusStorage _storage, uint _eventId)`

26. Multiple TODOs and unimplemented whitepaper features

It is expected that the Aventus development team will fix and implement the TODOs in the code. Features that were not yet implemented are: Deposit calculation, Ticket prices, Categories, Revenue sharing and Secondary markets.

Client's Response

The Aventus team acknowledged the issue, informing the TODOs will be completed in future versions.

27. Ambiguity in hashing arguments for events

For hashing inputs where there are two dynamically sized inputs (e.g. strings) there is a possibility for some abiguity in the hashing function:
http://solidity.readthedocs.io/en/v0.4.21/abi-spec.html#abi-packed-mode

Specifically, in the hashing inputs required to create a new event:
`LEvents.sol:353:function hashEventParameters(string _eventDesc, uint _eventTime, uint _capacity, uint _averageTicketPriceInUSCents, uint _ticketSaleStartTime, string _eventSupportURL, address _owner)`
there are two dynamically sized elements `_eventDesc` and `_eventSupportURL` which means there is some ambiguity in the construction of the hash (i.e. two different sets of inputs to `keccak256` could result in the same set of input packed data).

AMENDED [2018-05-22]:

The issue has been fixed by the Aventus team and is no longer present in commit `0xfcf572fbd48b76b5c90b2c4ff12c7fcc6e5eb903.`

Closing Summary

Beyond the issues mentioned, the contracts were also checked for overflow/underflow issues, DoS, and re-entrancy vulnerabilities. None were discovered. The code was found to be well tested for many different scenarios.

Code coverage was checked for the test cases provided with the contracts, and found to be generally high, although with some small gaps. The coverage report can be found at https://drive.google.com/open?id=1DatydWC2pkTZyzYE7PN0avVBIhbL3-6I

Disclaimer

Audit Report for Aventus Protocol. May 11, 2018.

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.