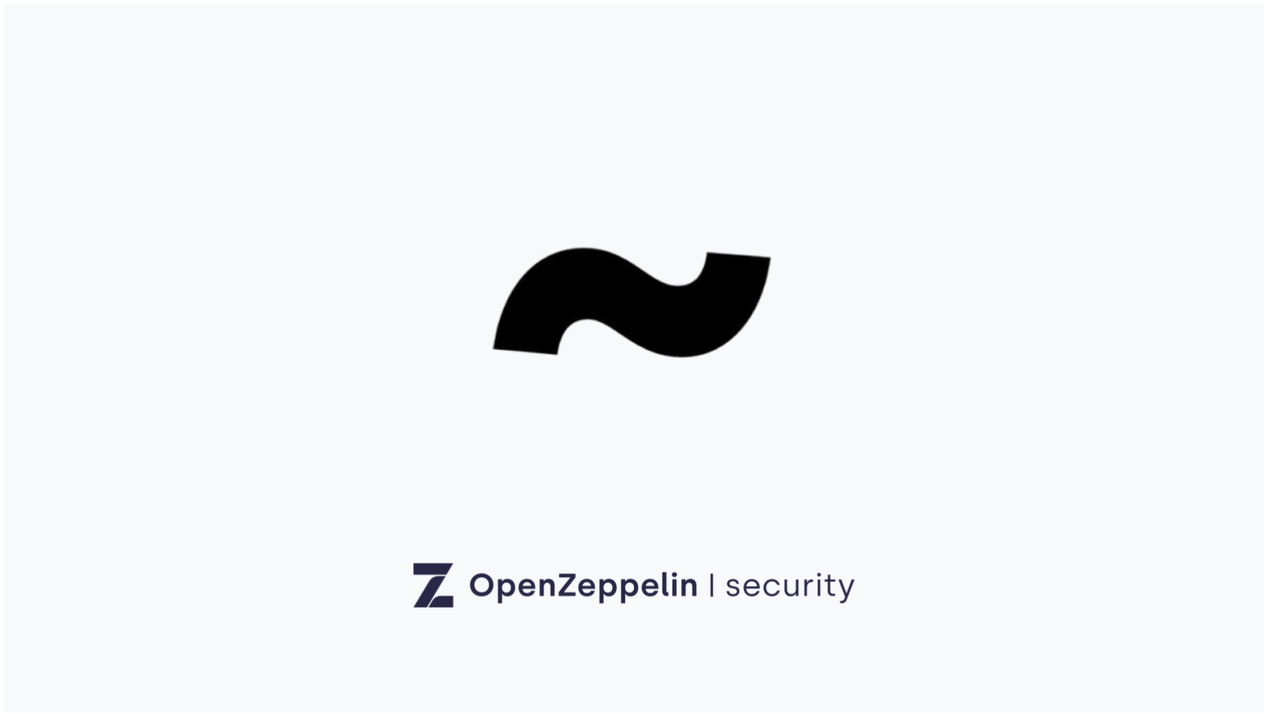
[DRAFT] Urbit Constitution Audit

SEPTEMBER 28, 2021 | IN PRIVATE REPORTS | BY OPENZEPPELIN



The Urbit team asked us to review and audit their Constitution contracts. We looked at the code and now publish our results.

The audited code is located in the urbit/constitution repository. The version used for this report is commit `55873ab559e61dd3de50289a6498eeba94a29605`.

Here is our assessment and recommendations, in order of importance.

Update: *The Urbit team has followed most of our recommendations and updated the contracts. The new version is at commit* `565331c88408f99573e225827ca9230c0ea2b9b8`.

Critical Severity

No issues of critical severity.

High Severity

No issues of high severity.

Medium Severity

Any launched ship can be adopted by ship 0

At creation of a ship in launch, its `escape` property is not set, hence it has the default value of `0`. This default behavior enables the ship `0` to adopt any newly launched ship without the pilot's permission.

Given that the signal for non-escaping ships is for its `escape` property to hold the value `65536`, consider configuring newly launched ships in this way.

Update: *Fixed in this commit by replacing the special 65536 value with a boolean escaping variable.*

Low Severity

Ships contract delegates lifecycle to its Constitution

The lifecycle of each ship in the Ships contract is composed of three stages: it starts with `Latent`, then `Locked` and finally `Living`. They can only transition to the next one.

The current implementation of the contract relies entirely on its `Constitution` to manage this lifecycle. For example, the start function checks if the `Locked -> Living` transition is valid by requiring the ship's state to be `Locked` and its lock time fulfilled.

This delegation seems safe under the current `Constitution` contract and makes sense if the goal is to allow more complex lifecycle rules in the future, like locking a `Living` ship under certain circumstances or starting a `Latent` ship. But given its upgradable nature, nothing prevents a future constitution to perform nonsensical lifecycle changes like trying to call setLiving on a living ship or trying to lock an already locked ship, emitting spurious `ChangedStatus` events.

Consider adding some basic lifecycle checks in the `Ships` contract to ensure its consistency.

Update: *Checks added in this commit.*

Unnecessarily large integer type in Hull

The doEscape function of the `Ships` contract requires the escape property of the fleeing ship to be lower than `65536` (which is 2^{16}). This implies that only stars or galaxies can adopt a ship. This is also enforced in the adopt and escape functions of the Constitution contract, in which the adopting ship is of type `uint16`.

The escape property of the `Ships` contract, however, is of type `uint32`, as well as the argument to setEscape, and it is possible to set the escape property of a ship to a `uint32` value greater than `65535`. Indeed, the value `65536` is used to signify that there is no escape active.

While the current Constitution interface properly uses only 16 bit values for escapes, through upgrading to a new constitution this could be changed. We would recommend to use a `uint16` type for the escape property of `Hull`, with an additional boolean to signify if there is an escape active. Such types would be more representative of the actual semantics of the `Hull` struct.

Update: *Fixed in this commit by changing the _parent variable type to uint16.*

Notes & Additional Information

Constitution contract

- According to revokeLaunchRights's documentation, the only requirement to execute the function is for the ship's owner to do it. Instead, the contract's code also requires the ship to be alive. If this is intended, consider updating the documentation. (***Update:*** *Documentation updated in this commit.)*

Ships contract

- The `getOriginalParent` function is defined with the `constant` modifier, which promises to not write in storage. Consider changing it for the `pure` modifier as it also promises to not read from it. (***Update:*** *Fixed in this commit.)*
- The `getChildren` function return type is `uint32` but the `children` property is defined as `uint16`. Consider updating the function's return type in order to make them match. (***Update:*** *Fixed in this commit.)*

Votes contract

- The `castConcreteVote` function does not have an explicit return value if the vote is negative. Consider adding one in order to be explicit about the function semantics. (***Update:*** *Fixed in this commit.)*

SafeMath8 contract

- The `SafeMath8` contract requires Solidity's version to be `^0.4.11` in contrast with the rest of the contracts which requires `0.4.18`. Consider upgrading it to match the rest of the contracts. (***Update:*** *Fixed in this commit.)*

Conclusion

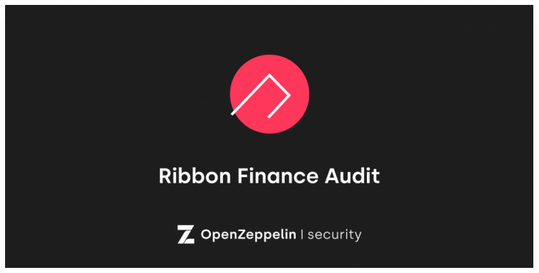
No critical severity or high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

We would also like to mention the good health of the codebase in terms of clarity, encapsulation and documentation.

If you are interested in discussing smart contract security, join our slack channel, follow us on Medium, or apply to work with us! We are also available for smart contract security development and auditing work.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to Urbit's Constitution contracts. We have not reviewed the related Urbit project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

RELATED POSTS

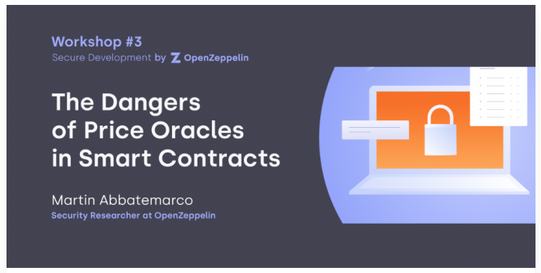


SECURITY AUDITS

Ribbon Finance Audit

The Ribbon Finance team asked us to review and audit their Theta Vault and Delta Vault smart...

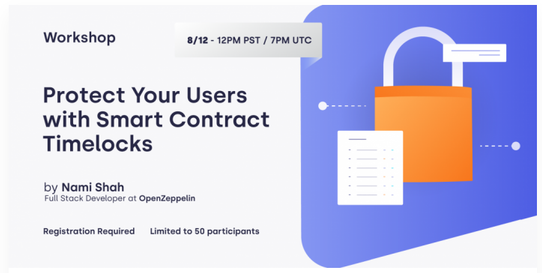
READ MORE



EVENTS GUIDELINES

Smart Contract Security Guidelines #3: The Dangers of Price Oracles

This guide focuses on showcasing the architecture, roles and subtleties of most



EVENTS

Protect Your Users With Smart Contract Timelocks

Learn how to use OpenZeppelin's Defender to set up and deploy timelocks.

READ MORE

popular price...

READ MORE

OpenZeppelin

Email*

Get our monthly news roundup

protected by reCAPTCHA
Privacy - Terms

SIGN UP