SOLIDIFIED

Audit Report for Xaya Democrit - May 30, 2023

Summary

Audit Report prepared by Solidified covering the Xaya Democrit contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on May 24, 2023, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repositories:

Repo: https://github.com/xaya/delegation-contract
Commit hash: `6395df328834632154e837199fb62e11896e3de4`
Fixes received at commit `9b69c659dbe0ebcc2525135456338b047820807a`

```
contracts
├── JsonSubObject.sol
├── JsonSubObjectTestHelper.sol
├── MovePermissions.sol
├── MovePermissionsTestHelper.sol
├── NamePermissions.sol
├── UnsafeForwarder.sol
└── XayaDelegation.sol
```

Repo: https://github.com/xaya/democrit-evm
Commit hash: `b9875a762330e5497afa0ce95a5938516d0176bb`
Fixes received at commit `b29b85788cb46f5ef9ef1411f69ad4188c543e3d`

```
contracts
├── AccountHolder.sol
├── AccountHolderTestHelper.sol
├── Democrit.sol
├── DemocritTestHelper.sol
├── IDemocritConfig.sol
├── JsonUtils.sol
```

```
├── JsonUtilsTestHelper.sol
├── LimitBuying.sol
├── LimitSelling.sol
├── TestConfig.sol
├── VaultManager.sol
└── VaultManagerTestHelper.sol
```

Intended Behavior

The audited code base implements Xaya accounts delegation on EVM-based chains and atomic trading for Xaya applications.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	High	An extensive system documentation was provided. Also, the reasoning behind many implementation details is explained well in the codebase.
Test Coverage	High	-

Issues Found

Solidified found that the Xaya Democrit contracts contain no critical issues, no major issues, 1 minor issue, and 5 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Push pattern for AccountHolder initialization can lead to errors	Minor	Acknowledged
2	isSafe function misses to check right-to-left operator	Note	Acknowledged
3	Usage of _msgSender() in Democrit	Note	Acknowledged
4	Different pragma and openzeppelin-contracts versions	Note	Acknowledged
5	@truffle/hdwallet-provider not in delegation package.json	Note	Fixed
6	Usage of assert throughout the codebase	Note	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. Push pattern for AccountHolder initialization can lead to errors

The `AccountHolder` contract is initialized within `onERC721Received` when a Xaya account NFT is transferred to it. This hook is only executed when `safeTransferFrom` is used for transfers (but not for `transferFrom`). Therefore, the usage of `transferFrom` would lead to a situation where the NFT is locked up irrevocably, but the contract is not initialized.

Recommendation
Consider using a pull pattern for the initialization where an initializer that is explicitly called transfers the NFT from the caller. Like that, this error cannot happen.

Status
Acknowledged: “We decided to implement the push mechanics, since it requires only one transaction for initialisation (sending of the account name) vs two (ERC-721 approval and triggering the pull from the contract).

In our situation, we think the potential drawbacks and risks are minimal for the following reasons:

- Potential locking of NFTs is a risk with transferFrom and any NFT or contract, independent of this specific choice we made.
- The initialisation is something done only once and by us, not by ordinary users. If someone is forking our contract and deploying their own Democrit market, they should know what they are doing, too.
- The deployment, registration of an account, and initialisation with it is actually done by a deployment script, so there is not even a risk of messing it up manually by us.
- The account name NFT is not valuable at all, and can be replaced easily if something were to go wrong during initialisation."

Informational Notes

2. `isSafe` function misses to check right-to-left operator

The `isSafe` function in `delegation-contract/contracts/JsonSubObject.sol` ensures that the user-provided string cannot lead to an injection of JSON syntax that breaks out of the intended path it is placed at. However, the function does not check whether the Right To Left Operator is used inside the string which will render the string from right to left on the UI if any part of the JSON will ever be used on the UI.

Recommendation
Consider checking for characters like RTLO which can influence the display of the names.

Status
Acknowledged: "As you noted, the purpose of `JsonSubObject.isSafe()` is to ensure that no JSON injection attack can be done. The question about strange (but valid) Unicode strings, look-alike attacks and UI display is not within scope of this method. (And in fact, the strings on which it operates are only meant to be processed by the rollup logic and will not be displayed in any UI.) Thus we prefer to keep the scope as it is."

3. Usage of `_msgSender()` in Democrit

While the `_msgSender()` function offered by the Context contract can be valuable in specific situations, it is crucial to recognize its limitations and potential drawbacks. In the context of Democrit or similar cases where no special signing is used, opting for `msg.sender` directly can enhance code legibility. By avoiding the use of `_msgSender()`, the code becomes clearer and easier to comprehend, without the need for an additional function call.

Recommendation

Consider changing `_msgSender()` to `msg.sender` on the Democrit repository.

Status

Acknowledged: “We understand the points you make about msg.sender instead of `_msgSender()`. However, please note that the Democrit contract is just the reusable part of the base protocol, and the actual contract deployed will contain some project-specific logic added on top. In our case, we will most likely want to support EIP-2771 meta transactions through this contract, and thus the use of `_msgSender()` and the Context is required.”

4. Different pragma and openzeppelin-contracts versions

The pragma versions on both repositories differ, as well as the OpenZeppelin contract versions.

Recommendation

Consider upgrading the delegation-contract repository to pragma solidity 0.8.19 and OpenZeppelin 4.8.0

Status

Acknowledged: “We understand the suggestion you make, and believe that it is in general a good one. But in our case, we decided to keep the codebase as-is, for the following reason: The two contracts (delegation and Democrit) are in principle separate from each other, except

that Democrit is using delegation. We submitted them together for audit mainly for organisatorial reasons. Furthermore, the delegation contract is already deployed and in use by our running beta (where assets do not have any real value yet). We prefer to avoid redeploying it just for this one reason, which does not present any actual issue with the code or security risk to users."

5. `@truffle/hdwallet-provider` not in delegation package.json

The package `@truffle/hdwallet-provider` is required for the project but it is not present in the `package.json` file.

Recommendation
Consider adding this package to the `package.json` file

Status
Fixed

6. Usage of `assert` throughout the codebase

It is generally advised to use `require` instead of `assert` in Solidity smart contracts. `require` is intended for input validation and conditions that must be met for the function to execute successfully. Furthermore it does not consume all the remaining gas after it fails.

Recommendation
Consider changing `assert` for `require` throughout the codebase.

Status
Acknowledged: "Indeed, we use both `require()` and `assert()` in our code. `require()` is used for input checks, i.e. conditions that a user can trigger.
`assert()` is used for conditions that should always be true independent of user input, and whose failure would by itself indicate a severe bug in the code. This distinction makes the intent clear while reading the code, and also, for instance, allows us to properly interpret test coverage

(which is impossible to get on the "else" branches of assert conditions by definition). We prefer to keep the usage of `require()` and `assert()` as it is."

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Autonomous Worlds Ltd or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. 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.

Oak Security GmbH