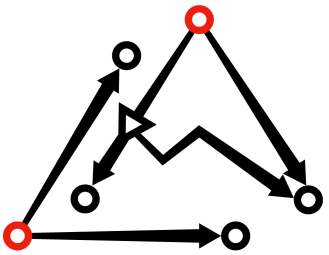

Least Authority

PRIVACY MATTERS

ERC-1155 Token Smart Contract

Security Audit Summary

Arkane Network

Final Audit Summary Report Version: 3 August 2021

Table of Contents

Overview

Background

Arkane Network has requested that Least Authority perform a security audit of their ERC-1155 Non-Fungible Token.

Project Dates

- **May 13 - 17**: Code Review *(Completed)*
- **May 18**: Initial Audit Summary Report delivered *(Completed)*
- **August 3**: Verification completed and Final Audit Summary Report delivered *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the ERC-1155 Token Smart Contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following is the MD5 hash of the source code:

`Ec6c6974f1190988682fab8a17d0e653`

For the verification, we examined:

`6f9d372e1701ff6101a459ee11813420`

Supporting Documentation

The following documentation was available to the review team:

- Open Zeppelin Documentation: https://docs.openzeppelin.com/contracts/4.x/erc1155

Areas of Concern

Our investigation focused on the following areas:

- Adherence to best practices and standards;
- Correctness of the implementation;
- Vulnerabilities in the smart contracts code;
- Common attacks on the smart contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds or disrupt execution of the smart contracts;
- Basic token economics are functional;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Arkane token is an ERC-1155 standard multitoken. These tokens allow for a mapping of identifiers to balances, where the identifiers can represent many different tokens. The Arkane multitoken follows every standard and is well coded.

Attention to security is apparent, as the code uses standard extensions to help improve the ERC-1155 specification implementation. The code uses the latest Solidity compiler and makes use of the latest features, such as the new gas minimizing safe math. The following list of extensions are used on this multitoken:

- Context, for maintaining `msg.sender`;
- ERC-165, for ensuring smart contracts implement desired features;
- AccessControl, for permissioning functions clearly;
- Initializable, for single use functions;
- Domain separator in ERC-712 typed data, for meta transactions replay protection;
- Address library with isContract, for some probability that an address is an EOA or not; and
- Receiver checks, for handling the receipt of ERC-1155 transfers.

The inclusion of some extensions implies certain use cases that were out of scope for this security review. We suggest that documentation and testing be provided, facilitating a better understanding of the use cases of this multitoken.

We did not identify any security critical issues while reviewing the code and have made several best practice suggestions. We commend the Arkane team on their attention to detail and the use of extensions developed to enrich the ERC-1155 standard implementations.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Supply Documentation And Tests	Unresolved
Suggestion 2: Update To Latest Access Control Smart Contract	Resolved
Suggestion 3: Update To Latest Initializable Smart Contract	Resolved
Suggestion 4: Use OpenZeppelin Meta Transactions Implementation	Resolved
Suggestion 5: Consider Downgrade Of Solidity Compiler	Resolved

Suggestions

Suggestion 1: Supply Documentation And Tests

Synopsis

Generally, it may not be helpful to review tests or documentation for standard tokens as there is little variance in the review code and the standard implementations that have many tests elsewhere. While the Arkane tokens are standard, we noticed that it does contain some specialized code for which having documentation and tests would be useful. In this case there is code that implies that this will be used for a layer 2 network, but this can't be confirmed from the view of the code entirely. This is simply to inform the auditor of the use case of the tokens.

Mitigation

We recommend providing a simple description document for the use case of the Arkane tokens and minimal testing of the tokens functionality that the team feels is valuable, if any.

Status

The Arkane team has not provided additional tests or documentation at the time of verification. As a result, the suggestion remains unresolved.

Verification

Unresolved.

Suggestion 2: Update To Latest Access Control Smart Contract

Location

`AccessControl.sol`

`AccessControlMixin.sol`

Synopsis

The latest access control smart contracts have the `only role` modifier. Using this rather than the current mixin smart contract could remove the necessity of the mixin smart contract entirely. This will also give a standard error string which is more specific than the current one.

Mitigation

We recommend considering upgrading some of the extensions to the latest OpenZeppelin versions.

Status

The Akrane team is now using the latest and standard access control supplied by OpenZeppelin.

Verification

Resolved.

Suggestion 3: Use Latest Initializable Smart Contract

Location

`Initializable.sol`

Synopsis

As with Suggestion 2, there is a newer version of the OpenZeppelin initializable smart contract that provides some extra boolean checks. This smart contract is intended to be used for proxy smart contracts that cannot have a constructor, but can also be used in other situations where it is necessary to securely initialize outside of the constructor. There are no known security issues with the current initializer smart contract, we suggest examining the latest version for any useful features in the Arkane tokens use case.

Mitigation

We recommend examining if the latest initializer code is desired.

Status

The Akrane team is now using the latest and standard version of the OpenZeppelin initializable smart contract.

Verification

Resolved.

Suggestion 4: Use OpenZeppelin Meta Transactions Implementation

Location

`NativeMetaTransaction.sol`

Synopsis

OpenZeppelin now provides an implementation of meta transactions. Previously there were not any standard implementations and we have seen many functionally equivalent versions. We found no security concern with the version used by Arkane. However, the OpenZeppelin version includes useful checks (e.g. ensuring the forwarder has enough gas to complete the transaction).

Mitigation

We recommend using the OpenZeppelin implementation which includes checks. Additionally, standardized implementations are well tested and audited, reducing the risk of unknown vulnerabilities.

Status

The Arkane team has decided to keep their implementation of meta transactions. Given that there are no apparent flaws in their implementation and the state of meta transactions at this time, we consider this to be resolved.

Verification

Resolved.

Suggestion 5: Consider Downgrade Of Compiler

Synopsis

The Arkane tokens use the latest Solidity compiler version 0.8.4. At the time of writing this report, this compiler version is 24 days old. Generally, we advise against using the very latest version as there is a higher potential for unknown bugs. We do not have any reason to believe that there are critical issues in version 0.8.4, aside from the general history of issues arising.

Mitigation

We recommend using the 0.8.0 compiler version to avoid unknown risks from the latest version.

Status

Two compiler versions have been released since the beginning of this audit giving 0.8.4 enough time to mature. As a result, we no longer consider this suggestion to be valid.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, adherence to standards and extensions that have been developed by known teams and used in previous projects, and general common best practices of Solidity development. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.