SOLIDIFIED

Audit Report for Ivy Cash - July 28, 2021

Summary

Audit Report prepared by Solidified covering the Ivy Cash / Hawk smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 27 May 2021.

An extension audit was then performed and concluded on 28 July 2021.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/Waly-Cash/hawk/tree/audit-updates

Commit number: `dcf26b34e227ee46e3deef0c406285fec079904a`

The scope of the audit was limited to the following files:

```
contracts
├── EscrowManager
│   ├── ERC721Received.sol
│   ├── EscrowManager.sol
│   └── EscrowStruct.sol
├── DEXManager.sol
├── EscrowAdmin.sol
├── FeeManager.sol
├── Paymaster.sol
└── PriceManager.sol
```

Intended Behavior

The smart contracts implement an Escrow solution supporting a number of ERC20 and ERC721 tokens.

Code Complexity and Test Coverage

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 equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium	-
Test Coverage	High	-

Issues Found

Solidified found that the Ivy Cash contracts contain no critical issue, 2 major issues, 1 minor issue in addition to 2 informational notes.

1 Warning aimed at end users has been noted.

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

Issue #	Description	Severity	Status
1	Paymaster.sol: GSN integration allows malicious actors to waste Paymaster GSN Relay balance on failing transactions	Major	Pending
2	Paymaster.sol: function sellTokenForGas() can be called by anyone	Major	Pending
3	Tokens with optional return values are not supported	Minor	Pending
4	ERC20 tokens which charge transfer fees are not supported	Note	-
5	Miscellaneous notes	Note	-

Critical Issues

No critical Issues found.

Major Issues

1. Paymaster.sol: GSN integration allows malicious actors to waste Paymaster GSN Relay balance on failing transactions

A malicious actor can easily cause the `Paymaster.postRelayedCall()` hook to revert by not having enough Service Fee Tokens to cover for the gas costs.

In case when `Paymaster.postRelayedCall()` reverts, the GSN Relay will revert the user transaction, but will still charge the `Paymaster` for the costs of the failed transaction.

For a malicious actor such transactions would have zero costs.

Recommendation
Consider reviewing the integration with GSN network so that a malicious actor would not be able to cause the `Paymaster.postRelayedCall()` to revert without paying for it.

2. Paymaster.sol: function sellTokenForGas() can be called by anyone

While executing a relayed GSN transaction (for example `EscrowManager.withdrawAll()`) a malicious caller could invoke the `Paymaster.sellTokenForGas()` function to change the `latestFee` value in `Paymaster` (which was previously set by `EscrowManager._payFees()` function).

This allows the contract
- pay for transaction using a non-whitelisted `_serviceFeeToken`
- cause `Paymaster.postRelayedCall()` to revert thus incurring the failed transactions costs on `Paymaster`

Recommendation
Consider restricting `Paymaster.sellTokenForGas()` to be called only by `EscrowManager`.

Minor Issues

3. Tokens with optional return values are not supported

The contracts EscrowManager and DEXManager expects the token transfer and transferFrom methods to always return a value or revert on failure. This implementation will always fail with tokens where these methods return nothing, resulting in locked funds.

Recommendation
Cover all edge cases while implementing token transfer. OpenZeppelin's SafeERC20 library provides a convenient wrapper for this functionality: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol

Informational Notes

4. ERC20 tokens which charge transfer fees are not supported

The contracts EscrowManager and DEXManager do not check if the amount received is the same as the amount accounted for. This will work in almost all cases without any issues, except with some rare tokens which change a transfer fee.

Recommendation
It is recommended to inform the users about the incompatibility.

5. Miscellaneous notes

Miscellaneous notes for improving the code quality and readability.

- `EscrowManager.sol` & `DEXManager.sol`: Unused imports.
- `PriceManager.sol` & `DEXManager.sol`: No need to use `SafeMath` for `uint256`.

- `EscrowAdmin.sol` misspelled function name `isAcceptedServieFeeToken()`.
- `Paymaster.sol`: Change the `convertAmountOutToAmountInMaximum` method mutability to `pure`.

Recommendation
Consider updating the code with the notes.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Ivy Cash 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.

Solidified Technologies Inc.