By
Quit

Protocol Audit
Issue date
12/08/2022

Overview

The following is a review of the ApyMon Vault Contracts, performed by 0xQuit. ApyMon vaults are meant to be secure storage vaults for users, where they can store a variety of supported assets.

Supported assets include:

- ➔ ERC20 Tokens
- ➔ ERC721 Tokens
- ➔ ERC1155 Tokens
- ➔ Cryptopunks
- ➔ Ether

The purpose of this audit is to identify security vulnerabilities, potential gas savings, general best practice recommendations, as well as offer input from a design perspective to minimize community trust requirements.

This audit includes a best effort review of potential vulnerabilities, but it is not a guarantee of security. Findings should be used in conjunction with other audits (internal or external), as well as independent verification. Lastly, this audit is not an endorsement or dismissal of the product.

Thoughts

The contract is generally simple, which typically leaves little room for security flaws. The vaults can be deposited to and withdrawn from, and the Vault Key NFT acts as the key for creation of - and withdrawal from - a vault.

When evaluating this contract, the two main avenues of exploit I checked for were:

- ➔ Allowing unauthorized withdrawals
- ➔ Blocking authorized withdrawals
- ➔ Unauthorized movement of Vault Key NFTs

Findings

Ambiguous Key Contract

Severity: High Status: Resolved

- The team will be using the existing Apymon ERC721 contract as the key contract. This alleviates concerns about a potentially exploitable or malicious key contract: Apymon uses unaltered versions of

`safeTransferFrom`, `transferFrom`, `_transfer`, `setApprovalForAll`, and `getApproved`.

The repo uses a barebones ERC721 contract, but it is contained in a test folder which seems to indicate that it is not indicative of the key contract that will be used on mainnet. The logic that governs the key is paramount to the security of the resulting vault, so it is impossible to evaluate without that piece.

If the ERC721 contract used in tests does indeed reflect the mainnet implementation, then this is not a security concern. However, if that is the case, I would recommend using the Solmate implementation of ERC721, which adds several gas optimizations.

Use of Require Instead of Custom Errors
Severity: Gas Optimization
Strings are very expensive in solidity, which makes require(condition, error_string) less than ideal. Instead, opt to use custom errors which cut down on deploy and revert costs.

Extraneous Reentrancy Modifier
Severity: Gas Optimization
I don't see a reentrancy risk on `withdrawETH`. If the key owner were to re-enter, it would succeed, but they have permission to withdraw the full amount. If somebody other than the key owner were to attempt re-entry, the call would revert due to the `OnlyKeyOwner` modifier.

(Probably) Unnecessary SafeTransfer Usage
Severity: Gas Optimization
Since the destination should be known for all withdrawals, it would be cheaper to use `transferFrom` and `transfer` over their safe variants.

Use Pre-Increment Operator And Unchecked Blocks Where Possible
Severity: Gas Optimization

over/underflow is not a realistic concern in `Vault.withdrawMultiple`. Using a pre-increment operator and an unchecked block in the function's loop can save a small amount of gas.

No Deposit Function
Severity: QOL Recommendation
Users may find it less intimidating to deposit to the vault via a `deposit` function. This would of course add the step of requiring an approval, but may be a preferable option for some.

Recommend renaming `unlocked` to `onlyWhenUnlocked`
Severity: QOL Recommendation
`onlyWhenUnlocked` may be a better descriptor of what the modifier enforces.

Conclusion

No major issues were found and the vault can be considered reasonably safe. Several recommendations were made that can improve gas efficiency or QOL for users, but no critical vulnerabilities were found to be present. This is contingent on the key contract closely resembling the test ERC721 contract contained in the repo. If the key contract were to allow unauthorized key movement, then the security of the vault would accordingly be compromised.