Security Assessment

Hypervisor

Jul 12th, 2021

Table of Contents

Summary

This report has been prepared for Visor Finance to discover issues and vulnerabilities in the source code of the Hypervisor project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Hypervisor
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/VisorFinance/hypervisor
Commit	99115e1a27fc834785b6194d1f34ce949fd85f41

Audit Summary

Delivery Date	Jul 12, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	2	0	0	0	2	0
Informational	8	0	0	0	8	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
HVF	Hypervisor.sol	b1e9baecc9afa9af8e2c3ee107103b056f5e905db2717a6d65a9057b1d4983f5
HFV	HypervisorFactory.sol	4b929e72b1c47f2064276760b3ac99baf4b517b92749bf736c8cf3b349c68945

Dependencies

There are a few depending injection contracts or addresses in the current project:

- `token0`, `token1`, and `pool` for the contract `Hypervisor`;
- `tokenA`, `tokenB`, and `uniswapV3Factory` for the contract `HypervisorFactory`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

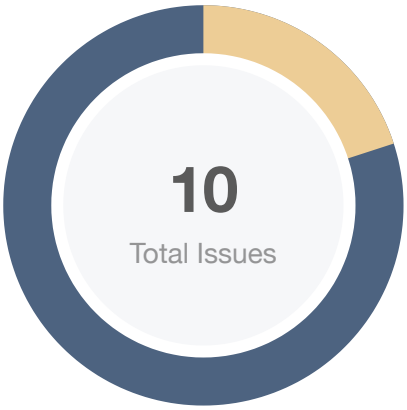
Privilledged Functions

The `owner` is an important role in the contract `Hypervisor`. The `owner` can operate on the following functions:

- `Hypervisor.setMaxTotalSupply()` to update the maximum of the total supply;
- `Hypervisor.setDepositMax()` to update the maximum amount of tokens to deposit in a single transaction.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**2** (20.00%)
Informational	**8** (80.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
HFV-01	Variable Could Be Declared as `immutable`	Gas Optimization, Language Specific	Informational	Acknowledged
HFV-02	Reentrancy Attack Risks	Logical Issue	Minor	Acknowledged
HVF-01	Unhandled Return Values	Coding Style	Informational	Acknowledged
HVF-02	Centralization Risks	**Centralization / Privilege**	**Informational**	**Acknowledged**
HVF-03	Missing Event Emissions for Significant Transactions	Coding Style	Informational	Acknowledged
HVF-04	Reentrancy Attack Risks	Logical Issue	Minor	Acknowledged
HVF-05	Incompatibility with Deflationary Tokens	Logical Issue	Informational	Acknowledged
HVF-06	`require` Statement Could Be Placed Before Calculation	Gas Optimization	Informational	Acknowledged
HVF-07	Edge Cases for Lower and Upper Bound	Coding Style	Informational	Acknowledged
HVF-08	Variables Could Be Declared as `immutable`	Gas Optimization, Language Specific	Informational	Acknowledged

HFV-01 | Variable Could Be Declared as `immutable`

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	contracts/HypervisorFactory.sol: 11	Acknowledged

Description

The state variable, `uniswapV3Factory`, that never changed after constructor can be declared as `immutable`.

Recommendation

We recommend declaring the aforementioned variable as `immutable`.

Alleviation

N/A

HFV-02 | Reentrancy Attack Risks

Category	Severity	Location	Status
Logical Issue	Minor	contracts/HypervisorFactory.sol: 25	Acknowledged

Description

The aforementioned function has external calls before state variable changes or event emissions. Thus, the function is vulnerable to reentrancy attacks.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

N/A

HVF-01 | Unhandled Return Values

Category	Severity	Location	Status
Coding Style	Informational	contracts/Hypervisor.sol: 102, 106, 201, 205	Acknowledged

Description

The functions `UniswapV3Pool.burn()` and `UniswapV3Pool.collect()` are not void-returning functions. Ignoring their return values, especially when their return values might represent the status of the transaction, might cause unexpected exceptions.

Recommendation

We recommend handling the return values of functions `UniswapV3Pool.burn()` and `UniswapV3Pool.collect()` before continuing processing.

Alleviation

N/A

HVF-02 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	contracts/Hypervisor.sol: 451, 457	**Acknowledged**

Description

The `owner` is an important role in the contract `Hypervisor`. The `owner` can operate on the following functions:

- `Hypervisor.setMaxTotalSupply()` to update the maximum of the total supply;
- `Hypervisor.setDepositMax()` to update the maximum amount of tokens to deposit in a single transaction.

Recommendation

We recommend the client carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt the `Timelock` contract with a reasonable delay to allow users to withdraw their funds, Multisig with community-selected 3-party independent co-signer, and/or DAO with transparent governance with the project's community in the project to manage the sensitive role accesses.

Alleviation

N/A

HVF-03 | Missing Event Emissions for Significant Transactions

Category	Severity	Location	Status
Coding Style	Informational	contracts/Hypervisor.sol: 451, 457	Acknowledged

Description

The functions that affect the status of sensitive variables should be able to emit events as notifications to the users. For example,

- `Hypervisor.setMaxTotalSupply()` to update the maximum of the total supply;
- `Hypervisor.setDepositMax()` to update the maximum amount of tokens to deposit in a single transaction;

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

Alleviation

N/A

HVF-04 | Reentrancy Attack Risks

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Hypervisor.sol: 90, 142, 185	Acknowledged

Description

The aforementioned functions have external calls before state variable changes or event emissions. Thus these functions are vulnerable to reentrancy attacks.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

N/A

HVF-05 | Incompatibility with Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Hypervisor.sol: 90, 142	Acknowledged

Description

The contract `Hypervisor` operates as the main entry for the interaction with the users. The users deposit token pairs and store them in a vault, and the token pairs are provided as liquidity using Uniswap V3. Later on, the users can withdraw their assets from the vault. In this process, `deposit()` and `withdraw()` may be involved in transferring users' assets into or out of the `Hypervisor`. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged (and burned) transaction fee. As a result, this may not meet the assumption behind these low-level asset-transferring routines and will bring unexpected balance inconsistency.

Recommendation

We recommend keeping regulating the set of tokens supported by the protocol, and if there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

Alleviation

N/A

HVF-06 | `require` Statement Could Be Placed Before Calculation

Category	Severity	Location	Status
Gas Optimization	Informational	contracts/Hypervisor.sol: 166	Acknowledged

Description

The `require` statement at the end of the function `Hypervisor.withdraw()` on L166 doesn't involve any variables that are calculated before the `require` statement, hence, moving the `require` statement to the place before all the computation is done could save gas in a case where the sender is not a valid token owner.

Recommendation

We recommend moving the `require` statement to the beginning of the function `Hypervisor.withdraw()` for gas optimization.

Alleviation

N/A

HVF-07 | Edge Cases for Lower and Upper Bound

Category	Severity	Location	Status
Coding Style	Informational	contracts/Hypervisor.sol: 193~196	Acknowledged

Description

When validating the tick ranges for the base position and the limit position, the lower tick should be greater than `TickMath.MIN_TICK`, and the upper tick should be less than `TickMath.MAX_TICK`.

Recommendation

We recommend adding the aforementioned validations to the tick ranges.

Alleviation

N/A

HVF-08 | Variables Could Be Declared as `immutable`

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	contracts/Hypervisor.sol: 32~36	Acknowledged

Description

State variables that never changed after constructor can be declared as `immutable`:

- `pool`,
- `token0`,
- `token1`,
- `fee`,
- `tickSpacing`.

Recommendation

We recommend declaring those aforementioned variables as `immutable`.

Alleviation

N/A

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.