Security Assessment

YF Farm

Apr 17th, 2021

Summary

This report has been prepared for YFVaults smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

It should be noted that the system design includes a number of economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself.

Additionally, financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The correctness of the financial model is not in the scope of the audit.

To bridge the trust gap between administrators and users, the administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- `governor` has the ability to alter key parameters in the strategy contract, including fee rates.
- `governor` has the ability to transfer `desire` and `earned` tokens to any address in case of trouble.

Findings

Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**5** (29.41%)
Informational	**12** (70.59%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
LSB-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
LSB-02	Lack of Input Validation	Volatile Code	Minor	Resolved
LSB-03	Redundant Check	Logical Issue	Informational	Resolved
SXB-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
SXB-02	Set `constant` to Variables	Logical Issue	Informational	Resolved
SXB-03	Incorrect Naming Convention Utilization	Coding Style	Informational	Resolved
SXB-04	Lack of Input Validation	Volatile Code	Minor	Resolved
SXB-05	Missing Emit Events	Gas Optimization	Minor	Resolved
SXB-06	Lack of Input Validation	Logical Issue	Minor	Resolved
SXB-07	Governor Capability	**Centralization / Privilege**	**Informational**	**Acknowledged**
SXB-08	Set `immutable` to Variables	Logical Issue	Informational	Resolved
YFP-01	Unlocked Compiler Version	Language Specific	Informational	Acknowledged
YFP-02	Set `constant` to Variables	Logical Issue	Informational	Resolved
YFP-03	Lack of input validation	Volatile Code	Minor	Resolved
YFP-04	Set `immutable` to Variables	Logical Issue	Informational	Resolved

LSB-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/LpSpell.sol: 2	Acknowledged

Description

The contract has unlocked compiler versions. An unlocked compiler version in the contract's source code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

LSB-02 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	contracts/LpSpell.sol: 28, 29	Resolved

Description

The assigned values to `router` and `pool` should be verified as non-zero values to prevent being mistakenly assigned as address(0) in the `constructor` function.

Recommendation

Check that the addresses are not zero by adding the following checks in the `constructor` function.

require(router != address(0),"Zero address");

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

LSB-03 | Redundant Check

Category	Severity	Location	Status
Logical Issue	Informational	contracts/LpSpell.sol: 220	Resolved

Description

This `require` check is redundant because balance of any ERC20 token is always non-negative.

Recommendation

We advise the client to remove this check.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/StratX.sol: 2	Acknowledged

Description

The contract has unlocked compiler versions. An unlocked compiler version in the contract's source code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

SXB-02 | Set `constant` to Variables

Category	Severity	Location	Status
Logical Issue	Informational	contracts/StratX.sol: 35	Resolved

Description

The variable `USDT` is not changed throughout the smart contract.

Recommendation

We advise the client to set `USDT` as a constant variable.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-03 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	contracts/StratX.sol: 51, 56, 59	Resolved

Description

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Constants should be in UPPER_CASE_WITH_UNDERSCORES

In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable. Examples:

- `withdrawFeeMax`
- `controllerFeeMax`
- `buybackRateMax`

Recommendation

We recommend renaming these constants as:

- `WITHDRAW_FEE_MAX`
- `CONTROLLER_FEE_MAX`
- `BUY_BACK_RATE_MAX`

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-04 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	contracts/StratX.sol: 86, 91, 92, 391, 395	Resolved

Description

The assigned values to `YFToken`, `desire`, `earned`, `governor`, `treasury` and `custodian` should be verified as non-zero values to prevent being mistakenly assigned as address(0) in the `constructor` function.

Recommendation

Check that the addresses are not zero by adding the following checks in the `constructor` function.

require(YFToken != address(0), "Zero address");

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-05 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Minor	contracts/StratX.sol: 391, 395, 400, 404, 408	Resolved

Description

Function that affect the status of sensitive variables should be able to emit events as notifications to customers:

- `setGov()`
- `setFundsAccount()`
- `setBuybackRate()`
- `setWithdrawFee()`
- `setControllerFee()`

Recommendation

Consider adding events for sensitive actions, and emit them in the function like below:

```
event SetGov(address indexed gov);

function setGov(address governor_) external onlyGov {
    governor = governor_;
    emit SetGov(governor_);
}
```

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-06 | Lack of Input Validation

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StratX.sol: 400~402, 404~406, 408	Resolved

Description

The governor might, by accident, set fee rates too high that harvest distribution incurs overflow.

Recommendation

We advise the client adding checks to ensure the sum of fee rates is less than one like as follows:

require(controllerFee1 + controllerFee2 + controllerFee2 + buybackRate < 10000, "Overflow of Proportions!");
require(withdrawFee < 100000, "Fee rate overflow!");

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

SXB-07 | Governor Capability

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	contracts/StratX.sol: 422	**Acknowledged**

Description

The governor has the capability to transfer 'earned' or 'desire' tokens to any address through `inCaseTokenGetStuck()`. While this capability offers a solution to the potential predicament the function name suggests, it could also be abused to enrich a selected few. This presents an issue we are concerned about and on which possible precautions we would like to inquire about.

Alleviation

`YFVaults`:The fact that a (deployed) StratX holds no assets makes 'inCaseTokensGetStuck()' rather secure. The function may be useful to deal with rounding errors.

SXB-08 | Set `immutable` to Variables

Category	Severity	Location	Status
Logical Issue	Informational	contracts/StratX.sol: 36, 41~44	Resolved

Description

The variables `YFToken`, `desire` and `earned` are only changed once in the `constructor()` function.

Recommendation

We advise the client to set `YFToken`, `desire`, `token0`, `token1` and `earned` as immutable variables.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

YFP-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	contracts/YFPool.sol: 2	Acknowledged

Description

The contract has unlocked compiler versions. An unlocked compiler version in the contract's source code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

YFP-02 | Set `constant` to Variables

Category	Severity	Location	Status
Logical Issue	Informational	contracts/YFPool.sol: 92, 94, 95, 96	Resolved

Description

The variables `USDT`, `startBlock`, `YFTokenMaxSupply` and `YFTokenPerBlock` are not changed throughout the smart contract.

Recommendation

We advise the client to set `USDT`, `startBlock`, `YFTokenMaxSupply` and `YFTokenPerBlock` as constant variables.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

YFP-03 | Lack of input validation

Category	Severity	Location	Status
Volatile Code	Minor	contracts/YFPool.sol: 102	Resolved

Description

The assigned value to `YFToken` should be verified as a non-zero value to prevent being mistakenly assigned as address(0) in the `constructor` function.

Recommendation

Check that the address is not zero by adding the following check in the `constructor` function.

require(YFToken != address(0),"Zero address");

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

YFP-04 | Set `immutable` to Variables

Category	Severity	Location	Status
Logical Issue	Informational	contracts/YFPool.sol: 90, 91	Resolved

Description

The variables `caster` and `YFToken` are only changed once in the `constructor()` function.

Recommendation

We advise the client to set `caster` and `YFToken` as immutable variables.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

YFP-05 | Set `constant` to Variables

Category	Severity	Location	Status
Coding Style, Gas Optimization	Informational	contracts/YFPool.sol: 94	Resolved

Description

`starBlock` is unchanged throughout the contract.

Recommendation

We advise the client to set `startBlock` as a constant variable.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

YFP-06 | Repetitive Codes

Category	Severity	Location	Status
Coding Style	Informational	contracts/YFPool.sol: 337~348, 410~421, 480~491	Resolved

Description

These codes are essentially the same and could be reused.

Recommendation

We advise client to reuse these codes in a function.

Alleviation

The team heeded our advice and resolved this issue in commit : 2f130286db4510aeba5df19ee17c302153ad0d0c.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations, etc.

Logical Issue

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

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

Language Specific

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

Coding Style

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.