Security Assessment

Autofarm

May 4th, 2021

Summary

This report has been prepared for Autofarm 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.

Overview

Project Summary

Project Name	Autofarm
Platform	BSC
Language	Solidity
Codebase	https://github.com/autofarm-network/AutofarmV2_CrossChain
Commits	1. 3d0c829895e0e7bca57c2cd0250ef8903dbb6022 2. 88af5a072c9bdc61a3a3d2708a1bde06aeffb63e

Audit Summary

Delivery Date	May 04, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	11
Critical	0
Major	3
Medium	0
Minor	2
Informational	6
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
AFT	AutoFarmTimelock_RewardsDistributor.sol	0606b61f7d024093878c78242c8675245eb368ba7a3c32d6aca9a709e7378ab2
ASC	AutoSwap.sol	c7bf8e1658704233d00da59d07746a5e109488b3257420fc53d5db6ea49e1b45
SXA	StratX2_AUTO.sol	eb35b0b6687fd9c988b811802e451d9a8bf9403053b260f9e74fadc9ac7f5751

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

`wbnbAddress` for contract `TimelockController_RewardsDistributor`;

`wbnbAddress`, `govAddress`, `autoFarmAddress`, `AUTOAddress`, `wantAddress`, `token0Address`, `token1Address`, `earnedAddress`, `farmContractAddres`, `uniRouterAddress`, `rewardsAddress`, `buyBackAddress`, `earnedToAUTOPath`, `earnedToToken0Path`, `earnedToToken1Path`, `token0ToEarnedPath` and `token1ToEarnedPath` for contract `StratX2_AUTO`;

`inToken`, `outToken` and `calls[i].target` for `AutoSwap.swap()`.

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

To set up project correctly, improve overall project quality and preserve the upgradability, the following role, are adopted in the codebase:

`TIMELOCK_ADMIN_ROLE`, is adopted to set other roles in contract `TimelockController_RewardsDistributor`;

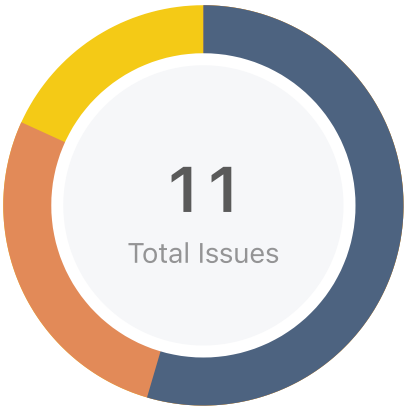
`PROPOSER_ROLE`, is adopted to schedule operations in contract `TimelockController_RewardsDistributor`;

`EXECUTOR_ROLE`, is adopted to execute operations in contract `TimelockController_RewardsDistributor`;

`_owner`, is adopted to update configurations in contract `AutoSwap`.

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

Findings



Critical	0 (0.00%)
Major	3 (27.27%)
Medium	0 (0.00%)
Minor	2 (18.18%)
Informational	6 (54.55%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
AFT-01	Variable Should Be Declared Constant	Language Specific	Informational	Acknowledged
AFT-02	Improper Usage of `public`	Gas Optimization, Language Specific	Informational	Acknowledged
ASC-01	Missing Check for Reentrancy Attack	Logical Issue	Major	Resolved
ASC-02	Missing Return Value Handling	Logical Issue	Minor	Partially Resolved
ASC-03	Misplaced Input Validation	Gas Optimization	Informational	Resolved
ASC-04	Unhandled Fees for Native Token	Logical Issue	Major	Resolved
ASC-05	Improper Usage of `public`	Gas Optimization, Language Specific	Informational	Acknowledged
SXA-01	Uninitialized Parameter	Logical Issue	Minor	Resolved
SXA-02	Missing Check for Size of Input Array	Logical Issue	Informational	Acknowledged
SXA-03	Missing Input Validation	Logical Issue	Informational	Acknowledged
SXA-04	Missing Check for Reentrancy	Logical Issue	Major	Resolved

AFT-01 | Variable Should Be Declared Constant

Category	Severity	Location	Status
Language Specific	Informational	AutoFarmTimelock_RewardsDistributor.sol: 110	Acknowledged

Description

Variable `wbnbAddress` is not modified within the contract and thus could be declared `constant`.

Recommendation

We recommend declaring `wbnbAddress` as `constant`.

Alleviation

N/A

AFT-02 | Improper Usage of `public`

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	AutoFarmTimelock_RewardsDistributor.sol: 266, 275, 316, 338, 391, 410, 432, 525, 556, 593, 621, 630, 634, 638, 642, 646, 654, 661, 668, 673, 680, 687, 694, 729	Acknowledged

Description

`public` functions that are never called within the contract should be declared `external`.

For example,

- `getTimestamp()`;
- `getMinDelay()`;
- `schedule()`;
- `scheduleBatch()`;
- `cancel()`;
- `execute()`;
- `executeBatch()`;
- `setDevWalletAddress()`;
- `scheduleSet()`;
- `executeSet()`;
- `add()`;
- `earn()`;
- `farm()`;
- `pause()`;
- `unpause()`;
- `rebalance()`;
- `deleverageOnce()`;
- `leverageOnce()`;
- `wrapBNB()`;
- `noTimeLockFunc1()`;
- `noTimeLockFunc2()`;
- `noTimeLockFunc3()`;
- `convertTokenToWBNB()`;
- `distributeRewards()`.

Recommendation

We recommend using the `external` attribute for functions never called within the contract.

Alleviation

N/A

ASC-01 | Missing Check for Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Major	AutoSwap.sol: 68	Resolved

Description

Function `swap` has state updates and event emits after external calls and thus is vulnerable to reentrancy attack.

Recommendation

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

Alleviation

The client heeded the advice and resolved this issue in commit `0f2dfac26ea22a6fba489f88fc23dd64d3cd8454`.

ASC-02 | Missing Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	AutoSwap.sol: 98	Partially Resolved

Description

Function `call()` is not a void-returning functions. Ignoring its return values, especially when its first return value represents the status if the transaction is executed successfully, might cause unexpected exceptions.

Recommendation

We recommend handling return values of function `call()` at the aforementioned line before continuing processing.

Alleviation

[Autofarm Team]: Contracts are trusted DEX routers which will revert upon failure.

ASC-03 | Misplaced Input Validation

Category	Severity	Location	Status
Gas Optimization	Informational	AutoSwap.sol: 189, 196	Resolved

Description

Used gas will not be refunded if `require()` triggers a revert. Therefore, input validations at the aforementioned lines should be placed at the beginning of functions for lower gas cost of failed calls.

Recommendation

We recommend performing input validations before executing other statements.

Alleviation

The client heeded the advice and resolved this issue in commit `0f2dfac26ea22a6fba489f88fc23dd64d3cd8454`.

ASC-04 | Unhandled Fees for Native Token

Category	Severity	Location	Status
Logical Issue	Major	AutoSwap.sol: 140	Resolved

Description

When `toToken` in function `_handleFees()` is the native token, i.e. `isETH()` returns `true` in library `UniversalERC20`, `universalTransfer` returns `false` when `amount` is non-zero. Then

- `referrerFee` will always be zero (line #169) even if `referrerFee` has been sent to `referrer` (line #165);
- `fee` will not be excluded from `outAmount` (line #174) even if `fee` has been sent to `owner()` (line #173).

Recommendation

The client heeded the advice and updated `UniversalERC20.universalTransfer()` in commit `f2178378e2d2af99db7b4f6126bf7a3ae96242d3` . Now `UniversalERC20.universalTransfer()` returns `true` upon non-zero transfers.

ASC-05 | Improper Usage of `public`

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	AutoSwap.sol: 68, 187, 194, 201, 205	Acknowledged

Description

`public` functions that are never called within the contract should be declared `external`.

For example,

- `swap()`;
- `changeFeeRate()`;
- `changeReferrerFeeRate()`;
- `pause()`;
- `unpause()`.

Recommendation

We recommend using the `external` attribute for functions never called within the contract.

Alleviation

N/A

SXA-01 | Uninitialized Parameter

Category	Severity	Location	Status
Logical Issue	Minor	StratX2_AUTO.sol: 10	Resolved

Description

Although parameter `minTimeToWithdraw` could be set by calling function `setMinTimeToWithdraw()`, it is is not initialized in `constructor()`, which means the contract will not be working properly after it is deployed without other settings.

Recommendation

We recommend setting `minTimeToWithdraw` in `constructor()`.

Alleviation

The client heeded the advice and resolved this issue in commit `88af5a072c9bdc61a3a3d2708a1bde06aeffb63e`.

SXA-02 | Missing Check for Size of Input Array

Category	Severity	Location	Status
Logical Issue	Informational	StratX2_AUTO.sol: 21	Acknowledged

Description

When the expected length of input array is pre-determined, checking the length of input array would be helpful to prevent potential human errors. In this case, length of `_addresses` for `constructor()` should be 12.

Recommendation

We recommend checking the length of input array `_addresses` before continuing processing.

Alleviation

N/A

SXA-03 | Missing Input Validation

Category	Severity	Location	Status
Logical Issue	Informational	StratX2_AUTO.sol: 26~34	Acknowledged

Description

Bounds of input parameters `_controllerFee`, `_buyBackRate`, `_entranceFeeFactor` and `_withdrawFeeFactor` are determined in `StratX2`:

- `_controllerFee <= controllerFeeUL`;
- `_buyBackRate <= buyBackRateUL`;
- `_entranceFeeFactor >= entranceFeeFactorLL` and `_entranceFeeFactor <= entranceFeeFactorMax`;
- `_withdrawFeeFactor >= withdrawFeeFactorLL` and `_withdrawFeeFactor <= withdrawFeeFactorMax`.

However, these checks are not performed in `constructor()`.

Also, addresses of input (first address in path) and output (last address in path) tokens in `_earnedToAUTOPath`, `_earnedToToken0Path`, `_earnedToToken1Path`, `_token0ToEarnedPath` and `_token1ToEarnedPath` should be checked.

Recommendation

We recommend performing the aforementioned checks in `constructor()`.

Alleviation

N/A

SXA-04 | Missing Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	Major	StratX2_AUTO.sol: 154	Resolved

Description

Function `earn()` has state updates and event emits after external calls and thus is vulnerable to reentrancy attack.

Recommendation

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

Alleviation

The client heeded the advice and resolved this issue in commit `31ae17ac24545d1aea585bffcca8708219ebdb09`.

Appendix

Finding Categories

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.

Mathematical Operations

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

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.

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

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.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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.