



Finify Protocol

Smart Contract Review

Deliverable: Smart Contract Audit Report

Security Report

June 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any person acting on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

© eNebula Solutions, 2021.

Report Summary

| | | | |
|---------------|-----------------------------------|---------------|------------|
| Title | Finify Smart Contract Audit | | |
| Project Owner | Finify Protocol | | |
| | | | |
| Type | Public | | |
| Reviewed by | Vatsal Raychura | Revision date | 29/06/2021 |
| Approved by | eNebula Solutions Private Limited | Approval date | 29/06/2021 |
| | | Nº Pages | 22 |

Overview

Background

FINIFY PROTOCOL requested that eNebula Solutions perform an Extensive Smart Contract audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

- **June 29:** Smart Contract Review Completed (*Completed*)
- **June 29:** Delivery of Smart Contract Audit Report (*Completed*)

Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of FINIFY PROTOCOL.

The following documentation repositories were considered in-scope for the review:

- FINIFY PROTOCOL Project:
<https://bscscan.com/address/0xdCB58F2DCE7613d833eF503767D8A1dBF68c9743#code>

Introduction

Given the opportunity to review FINIFY PROTOCOL Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About FINIFY PROTOCOL: -

| Item | Description |
|---------------------|---|
| Issuer | Finify Protocol |
| Website | https://finifyprotocol.finance/ |
| Type | BEP20 |
| Platform | Solidity |
| Audit Method | Whitebox |
| Latest Audit Report | June 29, 2021 |

The Test Method Information: -

| Test method | Description |
|-------------------|---|
| Black box testing | Conduct security tests from an attacker's perspective externally. |
| Grey box testing | Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses. |
| White box testing | Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc. |

Smart Contract Audit

The vulnerability severity level information:

| Level | Description |
|-----------------|---|
| Critical | Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities. |
| High | High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities. |
| Medium | Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities. |
| Low | Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed. |
| Weakness | There are safety risks theoretically, but it is extremely difficult to reproduce in engineering. |

The Full List of Check Items:

| Category | Check Item |
|------------------------------------|---------------------------------------|
| Basic Coding Bugs | Constructor Mismatch |
| | Ownership Takeover |
| | Redundant Fallback Function |
| | Overflows & Underflows |
| | Reentrancy |
| | MONEY-Giving Bug |
| | Blackhole |
| | Unauthorized Self-Destruct |
| | Revert DoS |
| | Unchecked External Call |
| | Gasless Send |
| | Send Instead of Transfer |
| | Costly Loop |
| | (Unsafe) Use of Untrusted Libraries |
| | (Unsafe) Use of Predictable Variables |
| | Transaction Ordering Dependence |
| | Deprecated Uses |
| Semantic Consistency Checks | Semantic Consistency Checks |
| | Business Logics Review |

Smart Contract Audit

| | |
|----------------------------|---|
| Advanced DeFi Scrutiny | Functionality Checks |
| | Authentication Management |
| | Access Control & Authorization |
| | Oracle Security |
| | Digital Asset Escrow |
| | Kill-Switch Mechanism |
| | Operation Trails & Event Generation |
| | ERC20 Idiosyncrasies Handling |
| | Frontend-Contract Integration |
| | Deployment Consistency |
| | Holistic Risk Management |
| Additional Recommendations | Avoiding Use of Variadic Byte Array |
| | Using Fixed Compiler Version |
| | Making Visibility Level Explicit |
| | Making Type Inference Explicit |
| | Adhering To Function Declaration Strictly |
| | Following Other Best Practices |

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

| Category | Summary |
|--|---|
| Configuration | Weaknesses in this category are typically introduced during the configuration of the software. |
| Data Processing Issues | Weaknesses in this category are typically found in functionality that processes data. |
| Numeric Errors | Weaknesses in this category are related to improper calculation or conversion of numbers. |
| Security Features | Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.) |
| Time and State | Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads. |
| Error Conditions, Return Values, Status Codes | Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function. |
| Resource Management | Weaknesses in this category are related to improper management of system resources. |

Smart Contract Audit

| | |
|-----------------------------------|---|
| Behavioral Issues | Weaknesses in this category are related to unexpected behaviors from code that an application uses. |
| Business Logics | Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. |
| Initialization and Cleanup | Weaknesses in this category occur in behaviors that are used for initialization and breakdown. |
| Arguments and Parameters | Weaknesses in this category are related to improper use arguments or parameters within function calls. |
| Expression Issues | Weaknesses in this category are related to incorrectly written expressions within code. |
| Coding Practices | Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained. |

Findings

Summary

Here is a summary of our findings after analyzing the FINIFY PROTOCOL 's Smart Contract. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

| Severity | No. of Issues |
|----------|--------------------------|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 1(Resolved/Acknowledged) |
| Total | 1 |

We have so far identified that there are potential issues with severity of **0 Critical, 0 High, 0 Medium, and 1 Low**. Overall, these smart contracts are well- designed and engineered, though the implementation can be improved and bug free by common recommendations given under POCs.

Functional Overview

| | |
|---------------------------|----------------|
| (\$) = payable function | [Pub] public |
| # = non-constant function | [Ext] external |
| | [Prv] private |
| | [Int] internal |

```
+ Context
- [Int] _msgSender
- [Int] _msgData

+ [Int] IERC20
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] transfer #
- [Ext] allowance
- [Ext] approve #
- [Ext] transferFrom #

+ [Lib] SafeMath
- [Int] add
- [Int] sub
- [Int] sub
- [Int] mul
- [Int] div
- [Int] div
- [Int] mod
- [Int] mod
```

- + [Lib] Address
 - [Int] isContract
 - [Int] sendValue #
 - [Int] functionCall #
 - [Int] functionCall #
 - [Int] functionCallWithValue #
 - [Int] functionCallWithValue #
 - [Prv] _functionCallWithValue #

- + Ownable (Context)
 - [Pub] <Constructor> #
 - [Pub] owner
 - [Pub] renounceOwnership #
 - modifiers: onlyOwner
 - [Pub] transferOwnership #
 - modifiers: onlyOwner
 - [Pub] getUnlockTime
 - [Pub] getTime
 - [Pub] lock #
 - modifiers: onlyOwner
 - [Pub] unlock #

- + [Int] IUniswapV2Factory
 - [Ext] feeTo
 - [Ext] feeToSetter
 - [Ext] getPair
 - [Ext] allPairs
 - [Ext] allPairsLength
 - [Ext] createPair #
 - [Ext] setFeeTo #
 - [Ext] setFeeToSetter #

```
+ [Int] IUniswapV2Pair
- [Ext] name
- [Ext] symbol
- [Ext] decimals
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] allowance
- [Ext] approve #
- [Ext] transfer #
- [Ext] transferFrom #
- [Ext] DOMAIN_SEPARATOR
- [Ext] PERMIT_TYPEHASH
- [Ext] nonces
- [Ext] permit #
- [Ext] MINIMUM_LIQUIDITY
- [Ext] factory
- [Ext] token0
- [Ext] token1
- [Ext] getReserves
- [Ext] price0CumulativeLast
- [Ext] price1CumulativeLast
- [Ext] kLast
- [Ext] burn #
- [Ext] swap #
- [Ext] skim #
- [Ext] sync #
- [Ext] initialize #

+ [Int] IUniswapV2Router01
- [Ext] factory
- [Ext] WETH
```

- [Ext] addLiquidity #
- [Ext] addLiquidityETH (\$)
- [Ext] removeLiquidity #
- [Ext] removeLiquidityETH #
- [Ext] removeLiquidityWithPermit #
- [Ext] removeLiquidityETHWithPermit #
- [Ext] swapExactTokensForTokens #
- [Ext] swapTokensForExactTokens #
- [Ext] swapExactETHForTokens (\$)
- [Ext] swapTokensForExactETH #
- [Ext] swapExactTokensForETH #
- [Ext] swapETHForExactTokens (\$)
- [Ext] quote
- [Ext] getAmountOut
- [Ext] getAmountIn
- [Ext] getAmountsOut
- [Ext] getAmountsIn

- + [Int] IUniswapV2Router02 (IUniswapV2Router01)
 - [Ext] removeLiquidityETHSupportingFeeOnTransferTokens #
 - [Ext] removeLiquidityETHWithPermitSupportingFeeOnTransferTokens #
 - [Ext] swapExactTokensForTokensSupportingFeeOnTransferTokens #
 - [Ext] swapExactETHForTokensSupportingFeeOnTransferTokens (\$)
 - [Ext] swapExactTokensForETHSupportingFeeOnTransferTokens #

- + FIN (Context, IERC20, Ownable)
 - [Pub] <Constructor> #
 - [Pub] name
 - [Pub] symbol
 - [Pub] decimals
 - [Pub] totalSupply

- [Pub] balanceOf
- [Pub] transfer #
- [Pub] allowance
- [Pub] approve #
- [Pub] transferFrom #
- [Pub] increaseAllowance #
- [Pub] decreaseAllowance #
- [Pub] isExcludedFromReward
- [Pub] totalFees
- [Pub] minimumTokensBeforeSwapAmount
- [Pub] buyBackUpperLimitAmount
- [Pub] deliver #
- [Pub] reflectionFromToken
- [Pub] tokenFromReflection
- [Pub] excludeFromReward #
 - modifiers: onlyOwner
- [Ext] includeInReward #
 - modifiers: onlyOwner
- [Prv] _approve #
- [Prv] _transfer #
- [Prv] swapTokens #
 - modifiers: lockTheSwap
- [Prv] buyBackTokens #
 - modifiers: lockTheSwap
- [Prv] swapTokensForEth #
- [Prv] swapETHForTokens #
- [Prv] addLiquidity #
- [Prv] _tokenTransfer #
- [Prv] _transferStandard #
- [Prv] _transferToExcluded #
- [Prv] _transferFromExcluded #

- [Prv] _transferBothExcluded #
- [Prv] _reflectFee #
- [Prv] _getValues
- [Prv] _getTValues
- [Prv] _getRValues
- [Prv] _getRate
- [Prv] _getCurrentSupply
- [Prv] _takeLiquidity #
- [Prv] calculateTaxFee
- [Prv] calculateLiquidityFee
- [Prv] removeAllFee #
- [Prv] restoreAllFee #
- [Pub] isExcludedFromFee
- [Pub] excludeFromFee #
 - modifiers: onlyOwner
- [Pub] includeInFee #
 - modifiers: onlyOwner
- [Ext] setTaxFeePercent #
 - modifiers: onlyOwner
- [Ext] setLiquidityFeePercent #
 - modifiers: onlyOwner
- [Ext] setMaxTxAmount #
 - modifiers: onlyOwner
- [Ext] setMarketingDivisor #
 - modifiers: onlyOwner
- [Ext] setNumTokensSellToAddToLiquidity #
 - modifiers: onlyOwner
- [Ext] setBuybackUpperLimit #
 - modifiers: onlyOwner
- [Ext] setMarketingAddress #
 - modifiers: onlyOwner

- [Pub] setSwapAndLiquifyEnabled #
 - modifiers: onlyOwner
- [Pub] setBuyBackEnabled #
 - modifiers: onlyOwner
- [Ext] prepareForPreSale #
 - modifiers: onlyOwner
- [Ext] afterPreSale #
 - modifiers: onlyOwner
- [Prv] transferToAddressETH #
- [Ext] <Fallback> (\$)

Detailed Results

Issues Checking Status

1. Message call with hardcoded gas amount

- SWC ID: 134
- Severity: Low
- Location: Finify.sol
- Relationships: CWE-655: Improper Initialization
- Description: Call with hardcoded gas amount. The highlighted function call forwards a fixed amount of gas. This is discouraged as the gas cost of EVM instructions may change in the future, which could break this contract's assumptions. If this was done to prevent reentrancy attacks, consider alternative methods such as the checks-effects-interactions pattern or reentrancy locks instead.
- Remediations: Avoid the use of `transfer()` and `send()` and do not otherwise specify a fixed amount of gas when performing calls. Use `.call.value(...)(")` instead. Use the checks-effects-interactions pattern and/or reentrancy locks to prevent reentrancy attacks.
- Acknowledged: After the first phase of Audit, This issue was discussed with the Finify Protocol's dev team, and they acknowledged this but as no serious or performance issue with this, they've decided to remain the code unchanged.

Basic Coding Bugs

1. Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: PASSED
- Severity: Critical

2. Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: PASSED
- Severity: Critical

3. Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: PASSED
- Severity: Critical

4. Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities
- Result: PASSED
- Severity: Critical

5. Reentrancy

- Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: PASSED
- Severity: Critical

6. MONEY-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: PASSED
- Severity: High

7. Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: PASSED
- Severity: High

8. Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: PASSED
- Severity: Medium

9. Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
- Result: PASSED
- Severity: Medium

10.Unchecked External Call

- Description: Whether the contract has any external call without checking the return value.
- Result: PASSED
- Severity: Medium

11.Gasless Send

- Description: Whether the contract is vulnerable to gasless send.
- Result: PASSED
- Severity: Medium

12.Send Instead of Transfer

- Description: Whether the contract uses send instead of transfer.
- Result: PASSED
- Severity: Medium

13. Costly Loop

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

14. (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

15. (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

16. Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

17. Deprecated Uses

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

Conclusion

In this audit, we thoroughly analyzed Finify Protocol 's Smart Contract. The current code base is well organized but there are promptly some low-level issues found in the first phase of Smart Contract Audit which is Acknowledged by the Finify Protocol's dev team afterwards and decided to remain unchanged as it won't create any security or performance issue in the contract.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting, please mail us at – contact@enebula.in