verichains

SECURITY AUDIT OF

FUNDGO ETF SMART CONTRACTS

Public Report

January 13, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on January 13, 2023. We would like to thank the TSS for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the FUNDGO ETF Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations. TSS team has resolved and updated all issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About FUNDGO ETF Smart Contracts 5

1.2. Audit scope 5

1.3. Audit methodology 5

1.4. Disclaimer 7

2. AUDIT RESULT 8

2.1. Overview 8

2.1.1. FundGoETFWrapped contract 8

2.1.2. FundCertMarketplace contract 9

2.2. Findings 9

2.2.1. Incorrect value for `unitTime` CRITICAL 10

2.2.2. Publisher can take advantage of `listP2P()` and `sellNow()` to drain contract funds CRITICAL 10

2.2.3. Contract logic can be disrupted by some functions MEDIUM 11

2.2.4. Incorrect check for order item in `deListingP2P` function LOW 11

2.3. Additional notes and recommendations 12

2.3.1. Inconsistent prices when selling tokens with `sellNow` function INFORMATIVE 12

2.3.2. Length of input arrays should be equal INFORMATIVE 13

2.3.3. Event missing indexed fields INFORMATIVE 13

2.3.4. Payment date can be controlled by users INFORMATIVE 14

3. VERSION HISTORY 15

1. MANAGEMENT SUMMARY

1.1. About FUNDGO ETF Smart Contracts

Digital Asset Management Center, or TSS for short, is the first unit in Vietnam to be recognized by the government as a legal entity responsible for promoting, organizing digitalization activities, conducting scientific research, and managing digital assets following the government's strategic direction regarding the digital economy.

FUNDGO Innovative Startup Investment Exchange Traded Fund (ETF) is a valuable security issued by the FUNDGO Innovative Startup Investment Fund in accordance with the fund's charter as regulated by the Law on support for small and medium enterprises. FUNDGO ETFs are considered valid investments in the fund, offering daily profitability, safety, easy transferability in times of need, and the opportunity to receive attractive profits for each term of the year as members of the fund.

With the establishment of the investment fund, the investment fund management company Trustpay must comply with strict regulations of the Goverment, so investors do not need to worry about their money being used for unclear purposes.

FUNDGO Innovative Startup Investment ETFs are digitized, stored, and authenticated using Blockchain technology provided by the trading platform of TSS - Digital Asset Management Center, which is established under the Vietnam Union of Science and Entrepreneurs.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of FUNDGO ETF Smart Contracts.

It was conducted on commit `6fa906d99552baa88cd0c5d2c5ecf524ac79ab34` from git repository link: *https://github.com/taisanso/etf*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`e2b936e05db6a252a23c1c02a2bbd610484077130936ea1c61a80387208aaa3f`	`etf.sol`
`3249136ac90b75c73ccd308b0fabd85064eeb67121c1def6875bdf10e189e0d1`	`etf-marketplace.sol`

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The FUNDGO ETF Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.9`. The source code was written based on OpenZeppelin's library.

2.1.1. FundGoETFWrapped contract

FundGoETFWrapped is a contract that follows the ERC1155 multi-token standard, implementing the issuance, selling, P2P transferring, harvesting, and redeeming of FUNDGO ETFs. This contract extends `ERC1155`, `Ownable`, `Pausable`, `ERC1155Burnable`, `ERC1155Supply`, and `ERC1155Receiver` contracts.

The direct token transfer functions are disabled by default, only transfers between `address(0x0)` and `address(this)` are accepted. The only whitelisted address that being allowed for calling the `customTransferFrom` function is the `FundCertMarketplace`. Additionaly, this contract does not support receiving ERC1155 batch tokens, as the `onERC1155BatchReceived` function has been disabled.

Only the contract owner can call the `setWithdrawer`, `emergencyWithdrawal`, `setMarketplace`, `pause`, `unpause`, `renounceOwnership` and `transferOwnership` functions, which are guarded by the `onlyOwner` modifiers.

Users of this contract can be distinguished as the publisher and normal users. The publisher is responsible for issuing the ETFs and withdrawing the remaining profit through the `withdrawIntervest` function. Normal users are permitted to trade their tokens among each other using the P2P functions: `listP2P`, `deListingP2P`, and `buyP2P`.

By holding the ETFs, user can claim their corresponding amount of profit in VNDC payment tokens based on the holding time. The holding time can span multiple continuous intervest terms, and the interest rate formula for each intervest term is as follows:

- Below 15 days: 6% (per year)
- From 15 days to below 30 days: 7%
- From 30 days to 92 days: 8%

User can choose to sell their ETFs at any time using the `sellNow` function and claim the according profit based on the formula above. These ETFs, along with their remaining profit, will be collected and claimed by the publisher later using the `withdrawIntervest` function.

Notice: The `withdrawer` account can call the `emergencyWithdrawal` function to withdraw funds from this contract. This address can be changed via the `setWithdrawer` function and can only be called by the contract owner (`onlyOwner`).

2.1.2. FundCertMarketplace contract

The FundCertMarketplace is a marketplace contract that allows publishers to sell their issued ETFs to users. This contract extends `Ownable`, `ERC1155Receiver`, and `ReentrancyGuard` contracts.

Only the contract owner can call the `addPublisher`, `removePublisher`, `setPaymentToken`, `setMarketStatus`, and `cleanMarket` functions, which are guarded by the `onlyOwner` modifiers.

This contract only allows whitelisted publishers to sell the ETFs to users via `sellItem`, `buyItem`, and `delistItem` functions. The price of the ETFs are determined by the `FundGoETFWrapped.getPriceAtTime(time)` function.

This contract also does not support receiving ERC1155 batch tokens, as the `onERC1155BatchReceived` function has been disabled.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of FUNDGO ETF Smart Contracts. TSS team updated the code, according to Verichains's draft report.

#	Issue	Severity	Status
1	Incorrect value for `unitTime`	CRITICAL	Fixed
2	Publisher can take advantage of `listP2P()` and `sellNow()` to drain contract funds	CRITICAL	Fixed
3	Contract logic can be disrupted by some functions	MEDIUM	Fixed
4	Incorrect check for order item in `deListingP2P` function	LOW	Fixed

Audit team also suggested some possible enhancements.

#	Issue	Status
1	Inconsistent prices when selling tokens with `sellNow` function	Acknowledged
2	Length of input arrays should be equal	Acknowledged
3	Event missing indexed fields	Acknowledged
4	Payment date can be controlled by users	Acknowledged

2.2.1. Incorrect value for `unitTime` CRITICAL

Affected files:

- etf.sol

The current value of `unitTime` is set to `1 minutes`, that will produce incorrect results when calculating `numberDays` in the `getPriceAtTime` function.

```
uint256 unitTime = 1 minutes;

function getPriceAtTime(uint256 _time) public view returns(uint256) {
    uint256 currentTime = _time;
    uint256 basePrice = etfInfor.price;
    for (uint256 index = 1; index <= numberTerm; index++) {
        uint256 startDate = etfInfor.issueDate + (etfInfor.intervestTerm * (index - 1) *
unitTime);
        uint256 endDate = etfInfor.issueDate + (etfInfor.intervestTerm * index * unitTime);

        if(currentTime > startDate && currentTime <= endDate){
            uint256 numberDays = (currentTime - startDate)/(unitTime); // INCORRECT
            return basePrice + (((basePrice * intervestTermRate)/baseRate)*numberDays)/365;
        }
    }

    return basePrice;
}
```

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged and fixed by the TSS team. The value of `unitTime` has been updated to `1 days`.

2.2.2. Publisher can take advantage of `listP2P()` and `sellNow()` to drain contract funds CRITICAL

Affected files:

- etf.sol

The publisher has the ability to repeatedly use the `sellNow()` function to sell their own tokens. Each time this function is called, the amount of `priceSellNow.price * amount` payment tokens may be sent to the publisher. As a result, the fund within the contract can be gradually drained by the publisher through multiple calls.

```
function sellNow(uint256 tokenId, uint256 amount)
    public
    whenNotPaused
    whenNotExpired
```

```
{
    // ...
    PriceSellNow memory priceSellNow = getPriceWhenSellNow(msg.sender, tokenId);
    totalAmount = (priceSellNow.price + priceSellNow.profit) * amount;
    uint256 parsedValue = etherFromWei(totalAmount);

    IERC20(paymentToken).safeTransfer(msg.sender, parsedValue); // FUND IN CONTRACT CAN BE
DRAINED
    // ...
}
```

Furthermore, the publisher can also sell his tokens using the `listP2P()` function, setting the price to zero, and transferring the tokens to his controlled account A. And he can later redeem these tokens, receiving an amount of `price * balance` payment tokens once again.

RECOMMENDATION

Do not allow the publisher to call `listP2P()` and `sellNow()` functions.

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged and fixed by the TSS team.

2.2.3. Contract logic can be disrupted by some functions MEDIUM

Affected files:

- etf.sol
- etf-marketplace.sol

Some of the functions, when called, can disrupt the logic of the contract such as:

- `FundCertMarketplace.setPaymentToken()`
- `FundGoETFWrapped.setIntervestTermRate()`

So, these functions should be used with caution. It is better to have some conditional statements that additionally guard their logic.

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged and fixed by the TSS team.

2.2.4. Incorrect check for order item in `deListingP2P` function LOW

Affected files:

- etf.sol

The conditional statement to check for the existence of the order is incorrect in the `deListingP2P` function. The condition `orderId > 0` should be replaced with `currentUserItem.orderId > 0`.

```
function deListingP2P(uint256[] memory orderIds) public whenNotPaused{
    address user = msg.sender;
    for (uint i = 0; i < orderIds.length; i++) {
        uint256 orderId = orderIds[i];
        NFTTradeP2P memory currentUserItem = findItemP2PItem(user, orderId);
        require(orderId > 0, "order not found"); // INCORRECT
        address otherUser = currentUserItem.user;
        uint256 tokenId = currentUserItem.tokenId;
        uint256 amount = currentUserItem.amount;
        bool isSelling = currentUserItem.isSelling;
        // ...
    }
    // ...
}
```

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged and fixed by the TSS team.

2.3. Additional notes and recommendations

2.3.1. Inconsistent prices when selling tokens with `sellNow` function INFORMATIVE

Affected files:

- etf.sol

In the `sellNow` function, users can sell their tokens to the publisher before the current intervest is ended. For example, if they buy their tokens 2 days after the `issueDate` (referred to as the `holdDate`) and then sell them after 3 days. The price of these tokens would be `price1 = getPriceAtTime(holdDate)`. However, if they choose to hold their tokens over the first intervest term and sell them 3 days after that, the price they would receive is `price2 = basePrice` as shown in the `getPriceWhenSellNow` function. It appears that users would lose some money if they hold their tokens for more than one intervest term.

```
function getPriceWhenSellNow(address _user, uint256 _tokenId) public view
returns(PriceSellNow memory) {
    // ...
    unchecked {
        uint256 etfTimeFromNow = (block.timestamp - issueDate)/unitTime;
        // case user sell nft in in term
        if((((holdDate - issueDate)/unitTime)/intervestTerm) ==
(etfTimeFromNow/intervestTerm)){
            uint256 price = getPriceAtTime(holdDate);
```

```
            holdingTime = (block.timestamp - holdDate)/(unitTime);
            rate = getIntervestRate(holdingTime);
            profit = (((basePrice * rate)/baseRate)*holdingTime)/365;
            actuallyAmount = price; // THE FIRST CASE
        }
        // case user sell nft in difference term
        else {
            uint256 startDate = issueDate + ((etfTimeFromNow/intervestTerm) *
intervestTerm) * (unitTime);
            holdingTime = (block.timestamp - startDate)/(unitTime);
            rate = getIntervestRate(holdingTime);
            profit = (((basePrice * rate)/baseRate)*holdingTime)/365;
            actuallyAmount = basePrice; // THE SECOND CASE
        }
        actuallyPaidProfit = (((basePrice * intervestTermRate)/baseRate)*holdingTime)/365;
        profitPublisher = actuallyPaidProfit - profit;
    }
    // ...
}
```

UPDATES

- *Jul 11, 2023*: This issue has been confirmed as intended by the TSS team.

2.3.2. Length of input arrays should be equal INFORMATIVE

Affected files:

- etf.sol

The length of `tokenIds` and `amounts` arrays should be equal in the following functions: `sellNow`, `listP2P`, and `redeem`. In these functions, the condition `require(tokenIds.length == amounts.length)` should be added in the beginning of the function logic.

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged by the TSS team.

2.3.3. Event missing indexed fields INFORMATIVE

Affected files:

- etf.sol
- etf-marketplace.sol

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use

three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

```
// etf.sol
event _eventListP2P(address _seller, address _buyer, uint256 _tokenId, uint256 _amount,
uint256 _totalValue, uint256 _orderId);
event _eventDeListP2P(address _seller, address _buyer, uint256 _tokenId, uint256 _amount,
uint256 _totalValue, uint256 _orderId);
event _eventBuyP2P(address _seller, address _buyer, uint256 _tokenId, uint256
_amount,uint256 _totalValue, uint256 _orderId, uint256 _holdingTime);
event _eventSellNow(address _seller, address _buyer, uint256 _tokenId, uint256 _amount,
uint256 _totalValue, uint256 _holdingTime);

// etf-marketplace.sol
event _eventListNewItemToMarket(uint256 id, uint256 nftId, uint256 amount, address seller);
event _eventDelistItem(uint256 id, uint256 nftId, uint256 amount, address seller);
event _eventBuyItemInMarket(uint256 id, uint256 nftId, uint256 amount, uint256 totalValue,
address seller, address buyer, address contractAddress);
```

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged by the TSS team.

2.3.4. Payment date can be controlled by users INFORMATIVE

Affected files:

- etf.sol

The `_paymentDate` input in the `payIntervest` and `payAdvanceIntervest` functions can be controlled by users. As a result, they can submit faked payment date and potentially abuse the value of `intervestHistory` mapping.

```
function payIntervest(uint256 _paymentDate) public {
    uint256 etherValue = etherFromWei(onceTermIntervest);
    IERC20(paymentToken).safeTransferFrom(msg.sender, etfInfor.publisher, etherValue);
    totalIntervestPayed += onceTermIntervest;
    intervestHistory[msg.sender][_paymentDate] = true;
}

function payAdvanceIntervest(uint256 _paymentDate) public {
    uint256 etherValue = etherFromWei(onceTermIntervest);
    IERC20(paymentToken).safeTransferFrom(msg.sender, address(this), etherValue);
    totalAdvanceIntervest += onceTermIntervest;
    intervestTempHistory[msg.sender][_paymentDate] = true;
}
```

UPDATES

- *Jul 11, 2023*: This issue has been acknowledged by the TSS team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 11, 2023*	Public Report	Verichains Lab

Table 2. Report versions history