Security Audit Report for Ferro Contracts

Date: April 28, 2022

Version: 1.0

Contact: contact@blocksecteam.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.04.28 20:41:30 +08'00'

Contents

Report Manifest

Item	Description
Client	Ferro
Target	Ferro Contracts

Version History

Version	Date	Description
1.0	April 28, 2022	First Released

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The files that are audited in this report include the following ones. The `Version 1` includes all files, and the others includes only files different from previous versions.

Version 1	
File Name	MD5
/ferro-buyback-audit/contracts/FerroMaker.sol	9e3c507ef753bedd4cf00402c1257176
/ferro-buyback-audit/contracts/StableSwapBurner.sol	93ed9ca05be04d8a483d5e7211d61d89
/ferro-buyback-audit/contracts/USDCBurner.sol	3c008bd5bd95f9156b6445b5fb6cc825
/ferro-farm-audit/contracts/FerroFarm.sol	46427baff937d9868b4393d226147db4
/ferro-farm-audit/contracts/FerroBoostDepositToken.sol	fc17d393519972a94874724d9dc85add
/ferro-farm-audit/contracts/FerroBoost.sol	0a17d168550127a60ab51ae604cce5b9
/ferro-farm-audit/contracts/FerroToken.sol	888e3b54708de84e23e0f1a80b7cf36a
/ferro-farm-audit/contracts/FerroVesting.sol	1359042ef79c3855f920de861492d264
/ferro-farm-audit/contracts/FerroBar.sol	42350495af965171d6f38910e363b301
/ferro-stableswap-master/contracts/AmplificationUtils.sol	6c19a718b3c00275460764d0b8a65b73
/ferro-stableswap-master/contracts/MathUtils.sol	e56a7a80beace24321fe317250b35e78
/ferro-stableswap-master/contracts/SwapDeployer.sol	66ec7e22b098159c60fd4d51670053b3
/ferro-stableswap-master/contracts/OwnerPausableUpgradeable.sol	976925fbd8d0125af93a3647540771b7
/ferro-stableswap-master/contracts/meta/MetaSwap.sol	04fbe511935cf421b8924210f963b993
/ferro-stableswap-master/contracts/meta/MetaSwapDeposit.sol	8b7cd1c9599a8c74e89ea7bbcbed10f6
/ferro-stableswap-master/contracts/meta/MetaSwapUtils.sol	61e73c3fd59b1dd88708883fbfb002eb
/ferro-stableswap-master/contracts/LPToken.sol	6066d5f2b55171ff6509699193b04ca1
/ferro-stableswap-master/contracts/Swap.sol	edddaff7c5fab26c43c68e43b815df66
/ferro-stableswap-master/contracts/helper/GenericERC20.sol	9cbd95a1484c6d9330a14297d986bccd
/ferro-stableswap-master/contracts/SwapUtils.sol	b573fcd2a568477df30b5c440644c799
Version 2	
/ferro-farm-audit/contracts/FerroBoost.sol	ee2126cb829f67dc9b64b8dd8dba196f
/ferro-farm-audit/contracts/FerroFarm.sol	bc60e8d70f00f361e022ed73ad865586
/ferro-farm-audit/contracts/FerroVesting.sol	d52864ce83c162dfd75a2b50b3875f57
/ferro-stableswap-master/contracts/meta/MetaSwap.sol	b22daf2126d3115761a63f4b38defc83
/ferro-stableswap-master/contracts/meta/MetaSwapUtils.sol	19dd5207ddcedb13737a0ec10e5bae7f
/ferro-stableswap-master/contracts/Swap.sol	ca720254961398d0919e5a6ce4bdeaaf
/ferro-stableswap-master/contracts/SwapUtils.sol	ea407b543c114567f97fcc44dde969d1

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [1] and Common Weakness Enumeration [2]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.

[1] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[2] https://cwe.mitre.org/

Table 1.1: Vulnerability Severity Classification

Impact	*High*	High	Medium
	Low	Medium	Low
		High	*Low*
		Likelihood	

- **Fixed** The issue has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **four** potential issues. We have **four** recommendations.

- High Risk: 0
- Medium Risk: 2
- Low Risk: 2
- Recommendations: 4

ID	Severity	Description	Category	Status
1	Low	The unnecessary check in *SwapUtils.sol* and *MetaSwapUtils.sol*	Software Security	Fixed
2	Medium	Reentrancy issues	DeFi Security	Fixed
3	Medium	Deflation token issue	DeFi Security	Fixed
4	Low	UUPS vulnerability	DeFi Security	Fixed
5	-	Fix incorrect comments	Recommendation	Fixed
6	-	Save gas	Recommendation	Fixed
7	-	Address the concern of the centralization design	Recommendation	Confirmed
8	-	Add a check for the functions `add` and `set` in the contract *FerroBoost*	Recommendation	Fixed

The details are provided in the following sections.

2.1 Software Security

2.1.1 The unnecessary check in *SwapUtils.sol* and *MetaSwapUtils.sol*

Severity Low

Status Fixed in `Version 2`.

Introduced by `Version 1`.

Description As shown in the following code snippet, the parameter `tokenAmount` is the amount of the lp token to burn. The function `calculateWithdrawOneTokenDY` calculates the number of tokens (indicated by `tokenIndex`) that could be withdrawn. However, the code in line 211 requires that the `tokenAmount` should be less or equal than `xp[tokenIndex]` that represents the pool balance of the token that will be withdrawn. This check seems to be a mistake.

```
219   function calculateWithdrawOneTokenDY(
220       Swap storage self,
221       uint8 tokenIndex,
222       uint256 tokenAmount,
223       uint256 totalSupply
224   )
225       internal
226       view
227       returns (
228           uint256,
229           uint256,
        uint256
    )
{
    // Get the current D, then solve the stableswap invariant
    // y_i for D - tokenAmount
    uint256[] memory xp = _xp(self);

    require(tokenIndex < xp.length, "Token index out of range");

    CalculateWithdrawOneTokenDYInfo
        memory v = CalculateWithdrawOneTokenDYInfo(0, 0, 0, 0, 0);
    v.preciseA = _getAPrecise(self);
    v.d0 = getD(xp, v.preciseA);
    v.d1 = v.d0.sub(tokenAmount.mul(v.d0).div(totalSupply));

    require(tokenAmount <= xp[tokenIndex], "Withdraw exceeds available");
```

Listing 2.1: SwapUtils.sol

This issue is also found in the function `_calculateWithdrawOneTokenDY` of `MetaSwapUtils.sol` (L263).

Impact The `removeLiquidityOneToken` will be reverted if the amount of the lp token to burn greater than the pool balance of the token to be withdrawn.

Suggestion Remove this check.

2.2 DeFi Security

2.2.1 Reentrancy issues

Severity Medium

Status Fixed in `Version 2`.

Introduced by `Version 1`.

Description There are a few functions that update critical variables after calling an untrusted token and use these variables before the untrusted call, which may be susceptible to the Re-entrancy attack. The details are shown in below code snippets.

```
102   function _withdraw(address _receiver) private {
103       User storage vestingMap = vestingInfo[_receiver];
104       uint256 amountToWithdraw = vestingAmount(_receiver) - vestingMap.released;
105       token.safeTransfer(_receiver, amountToWithdraw);
106       vestingMap.released += amountToWithdraw;
107       emit TokenReleased(_receiver, amountToWithdraw);
108   }
```

Listing 2.2: FerroVesting.sol

The amount of token to be withdrawn is calculated in line 104, which uses the variable `vestingMap.rel--eased`. If the token has a callback mechanism, then the Re-entrancy attack that delays the execution of code in line 106 and withdraws all the reserves of *FerroVesting* contract can work.

```
196   function deposit(uint256 _pid, uint256 _amount) external {
197       PoolInfo storage pool = poolInfo[_pid];
198       UserInfo storage user = userInfo[_pid][msg.sender];
199       updatePool(_pid);
200       if (user.amount > 0) {
201           uint256 pending = user.amount.mul(pool.accFerPerShare).div(1e18).sub(user.rewardDebt);
202           if (pending > 0) {
203               safeFerTransfer(msg.sender, pending, pool);
204           }
205       }
206       if (_amount > 0) {
207           pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
208           user.amount = user.amount.add(_amount);
209       }
210       user.rewardDebt = user.amount.mul(pool.accFerPerShare).div(1e18);
211       emit Deposit(msg.sender, _pid, _amount);
212   }
```

Listing 2.3: FerroFarm.sol

The code in line 210 updates the variable `user.rewardDebt` that is used to calculate the pending rewards. If the `pool.lpToken` has a callback mechanism, then the Re-entrancy attack can repeatedly obtain rewards by delaying the update of `user.rewardDebt`.

```
215   function withdraw(uint256 _pid, uint256 _amount) external {
216       PoolInfo storage pool = poolInfo[_pid];
217       UserInfo storage user = userInfo[_pid][msg.sender];
218       require(user.amount >= _amount, "withdraw: not good");
219       updatePool(_pid);
220       uint256 pending = user.amount.mul(pool.accFerPerShare).div(1e18).sub(user.rewardDebt);
221       if (pending > 0) {
222           safeFerTransfer(msg.sender, pending, pool);
223       }
224       if (_amount > 0) {
225           user.amount = user.amount.sub(_amount);
226           pool.lpToken.safeTransfer(address(msg.sender), _amount);
227       }
228       user.rewardDebt = user.amount.mul(pool.accFerPerShare).div(1e18);
229       emit Withdraw(msg.sender, _pid, _amount);
230   }
```

Listing 2.4: FerroFarm.sol

Similarly, if the `pool.lpToken`(in line 226) has a callback mechanism, then the Re-entrancy attack also works.

```
233   function emergencyWithdraw(uint256 _pid) external {
234       PoolInfo storage pool = poolInfo[_pid];
235       UserInfo storage user = userInfo[_pid][msg.sender];
236       pool.lpToken.safeTransfer(address(msg.sender), user.amount);
237       emit EmergencyWithdraw(msg.sender, _pid, user.amount);
238       user.amount = 0;
239       user.rewardDebt = 0;
240    }
```

Listing 2.5: FerroFarm.sol

The `emergencyWithdraw` function allows users to withdraw their deposits. If the `pool.lpToken` has a callback mechanism, the reset of `user.amount` (in line 238) can be delayed by the Re-entrancy attack.

Impact The reserves of *FerroFarm* and *FerroVesting* are at risk if they support tokens with callback mechanism, such as ERC777 tokens.

Suggestion Move the code updating critical variables before the untrusted call or use Re-entrancy guard. Otherwise, do not support tokens with a callback mechanism.

Feedback from the Project Not supporting token with callback mechanism. Only our stablecoin LP token.

For the contract FerroVesting, the project fixed the issue as our suggestion. For the contract FerroFarm, the project ensures that the `pool.lpToken` *has no callback mechanism. Therefore, we mark this issue as fixed.*

2.2.2 Deflation token issue

Severity Medium

Status Fixed.

Introduced by `Version 1`.

Description If the `pool.lpToken` is a deflation token that charges fees for each transfer, then the code in line 208 can not record the correct users' deposits.

```
196    function deposit(uint256 _pid, uint256 _amount) external {
197        PoolInfo storage pool = poolInfo[_pid];
198        UserInfo storage user = userInfo[_pid][msg.sender];
199        updatePool(_pid);
200        if (user.amount > 0) {
201            uint256 pending = user.amount.mul(pool.accFerPerShare).div(1e18).sub(user.rewardDebt);
202            if (pending > 0) {
203                safeFerTransfer(msg.sender, pending, pool);
204            }
205        }
206        if (_amount > 0) {
207            pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
208            user.amount = user.amount.add(_amount);
209        }
210        user.rewardDebt = user.amount.mul(pool.accFerPerShare).div(1e18);
211        emit Deposit(msg.sender, _pid, _amount);
212    }
```

Listing 2.6: FerroFarm.sol

Impact If the *FerroFarm* contract supports deflation token, each invocation of `deposit` and `withdraw` will cause the contract to lose assets.

Suggestion Record the balance change rather than the parameter `_amount`. Otherwise, do not support deflation tokens.

Feedback from the Project Not supporting token with deflation mechanism. Only our stablecoin LP token.

Since the project ensures that the `pool.lpToken` can not be deflation token, we mark this issue as fixed.

2.2.3 UUPS vulnerability

Severity Low

Status Fixed.

Introduced by `Version 1`.

Description The *FerroBoost* contract inherits the *UUPSUpgradeable* contract of OpenZeppelin. Before version 4.3.2 [1], the *UUPSUpgradeable* contract has a vulnerability that can cause the *FerroBoost* contract (logic contract) to self-destruct itself. The blog [2] gives a detailed description about the vulnerability.

Impact If the version of OZ library is less than 4.3.2, then it is possible to be attacked and cause the *FerroBoost* contract to self-destruct itself.

Suggestion Use a OZ library with a version higher than or equal to 4.3.2.

Feedback from the Project Lib is up to date. $^4.5.2$.

2.3 Additional Recommendation

2.3.1 Fix incorrect comments

Status Fixed in `Version 2`.

Introduced by `Version 1`.

Description There are some incorrect comments in codes:

- MetaSwap.sol(line 367): the ‘tokenAmount‘ should be "the amount of lp token to burn rather than the amount of the token you want to receive"
- MetaSwapUtils.sol(line 171): the ‘tokenAmount‘ should be "the amount of lp token to burn rather than the amount to withdraw in the pools precision"
- MetaSwapUtils.sol(line 227): the ‘toeknAmount‘ should be "the amount of lp token to burn" rather than "the amount to withdraw in the pools precision"
- Swap.sol(line 440): the ‘tokenAmount‘ should be "the amount of lp token to burn rather than the amount of the token you want to receive"
- SwapUtils.sol(line 139): the ‘tokenAmount‘ should be "the amount of lp token to burn rather than the amount to withdraw in the pools precision"
- SwapUtils.sol(line 190): the ‘tokenAmount‘ should be "the amount of lp token to burn rather than the amount to withdraw in the pools precision"

Impact NA

Suggestion Correct these comments.

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.3.2

[2] https://forum.openzeppelin.com/t/uupsupgradeable-vulnerability-post-mortem/15680

2.3.2 Save gas

Status Fixed in `Version 2`.

Introduced by `Version 1`.

Description There are recommendations that can save the gas usage.

- Remove the field `cliff` of the struct `User` in the contract *FerroVesting*, because it is not used.
- Modify the function `getMultiplier` of the contract *FerroFarm* as internal function, because it has no meaning to expose it to the user.
- Remove the usage of SafeMath library in the contract *FerroFarm*, because Solidity higher than 0.8.0 does the same checks for math operations.
- Make the variables `startBlock` and `MAX_FER_PER_BLOCK` in the contract *FerroFarm* as immutable variables, because it has no code to update them.
- Make the variables: `ferroFarm`, `fer`, `xFER`, and `depositToken` in the contract *FerroBoost* as immutable variables, because it has no code to update them.
- Make the variable `depositTokenPid` in the contract *FerroBoost* as constant 0.

Impact NA

Suggestion NA.

2.3.3 Address the concern of the centralization design

Status Confirmed.

Introduced by `Version 1`.

Description As shown in below code, the governance token FER can be minted by the owner of the contract *FerroToken*.

```
contract FerroToken is ERC20, Ownable {
    constructor() ERC20("FerroToken", "FER") {}

    /// @notice Creates ‘_amount‘ token to ‘_to‘. Must only be called by the owner (FerroFarm).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}
```

Listing 2.7: FerroToken.sol

Impact Authorized accounts can infinitely mint the governance token FER. If the private keys of these accounts are leaked, the project will collapse.

Suggestion Adopt a decentralized method to manage authority (e.g., DAO contract), or leverage a secure private key solution (e.g., multi-signed wallet, and TEE based security key management) to manage the private keys of authorized EOAs.

Feedback from the Project The token ownership will goes to FerroFarm – should be alright. Flow (similar to VVS): 1. create token 2. mint token to different wallet (based on tokenomic) 3. transfer ownership to farm

2.3.4 Add a check for the functions `add` and `set` in the contract *FerroBoost*

Status Fixed in `Version 2`.

Introduced by `Version 1`.

Description As shown in below code, the variable `pool.multiplier` must be greater than zero.

```
function depositFor(
    uint256 _pid,
    uint256 _amount,
    address _user
) public {
    PoolInfo storage pool = poolInfo[_pid];
    require(pool.multiplier > 0, "FerroBoost: Invalid Pool ID");
```

Listing 2.8: FerroBoost.sol

Impact NA.

Suggestion Add the check `require(_multiplier>0, "XXX")` for the functions `add` and `set`.