METATRUST

Security Assessment for

Accumulated Finance

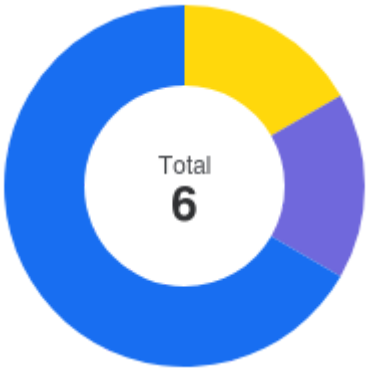
June 29, 2024

Executive Summary

Overview	
Project Name	Accumulated Finance
Codebase URL	https://github.com/AccumulatedFinance/contracts-v2
Scan Engine	Security Analyzer
Scan Time	2024/06/29 08:00:00
Commit Id	41ef980cc65a81b256ed7616e4f26d33597e72a8

Total	
Critical Issues	0
High risk Issues	0
Medium risk Issues	1
Low risk Issues	1
Informational Issues	4

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	0%	0
Medium risk Issues	17%	1
Low risk Issues	17%	1
Informational Issues	67%	4

Summary of Findings

MetaScan security assessment was performed on **June 29, 2024 08:00:00** on project **Accumulated Finance** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **6** vulnerabilities / security risks discovered during the scanning session, among which **1** medium risk vulnerabilities, **1** low risk vulnerabilities, **4** informational issues.

Besides the audited contract `Minter`, there are following two contracts in the Accumulatd Finance:

The `stToken` contract, it is an ERC20 token contract, and its owner has the privilege of the following functions:

- `pause`: Allows the owner to pause all token transfers;
- `unpause`: Allows the owner to unpause all token transfers;
- `mint`: Allows the owner to mint new tokens to a specified address.

The `wstToken` contract, it is a fork of the audited and deployed sfrxETH with the below little update:

- renamed sfrxETH to wstToken
- removed the `andSync` modifier from functions, `deposit`, `mint`, `withdraw`, and `redeem`.
- the `syncRewards` function reverts if the `nextRewards` is zero.

ID	Description	Severity	Alleviation
MSA-001	Centralization Risk	Medium risk	Acknowledged
MSA-002	The `minWithdrawal` lacks the upper boundry	Low risk	Fixed
MSA-003	Unused Return Value	Informational	Fixed
MSA-004	SafeMath Can be Remove on the Solidity Version 0.8.0 or Above 0.8.0	Informational	Fixed
MSA-005	The Redundant Check on `request.claimed` From the `processWithdrawals` Function	Informational	Fixed
MSA-006	No Need to Use `safeTransferFrom`	Informational	Fixed

Findings

Medium risk (1)

1. Centralization Risk

Medium risk | Security Analyzer

In the `BaseMinter` contract, the owner has the privilege of the following functions:

- `updateDepositFee`: Allows the owner to update the deposit fee up to a maximum of 5%;
- `transferStakingTokenOwnership`: Allows the owner to transfer the ownership of the staking token to a new owner;
- `mint`: Allows the owner to mint new staking tokens to a specified address.

In the `NativeMinter` contract, the owner has the privilege of the following functions:

- `withdraw`: Allows the owner to withdraw all the network coin balance from the contract to a specified address.

In the `ERC20Minter` contract, the owner has the privilege of the following functions:

- `withdraw`: Allows the owner to withdraw all the base token balance from the contract to a specified address.

In the `BaseMinterRedeem` contract, the owner has the privilege of the following functions:

- `updateRedeemFee`: Allows the owner to update the redeem fee up to a maximum of 5%.

In the `BaseMinterWithdrawal` contract, the owner has the privilege of the following functions:

- `updateWithdrawalFee`: Allows the owner to update the withdrawal fee up to a maximum of 5%;
- `updateMinWithdrawal`: Allows the owner to update the minimum withdrawal amount;
- `processWithdrawals`: Allows the owner to process multiple withdrawal requests;
- `collectWithdrawalFees`: Allows the owner to collect accumulated withdrawal fees from the contract.

In the `NativeMinterWithdrawal` contract, the owner has the privilege of the following functions:

- `withdraw`: Allows the owner to withdraw the available network coin balance from the contract to a specified address.

In the `ERC20MinterWithdrawal` contract, the owner has the privilege of the following functions:

- `withdraw`: Allows the owner to withdraw the available base token balance from the contract to a specified address.

File(s) Affected

Minter.sol #1949-1953

```
    function updateRedeemFee(uint256 newFee) public onlyOwner {
        require(newFee <= MAX_REDEEM_FEE, ">MaxFee");
        redeemFee = newFee;
        emit UpdateRedeemFee(newFee);
    }
```

Minter.sol #2065-2075

```
    function updateWithdrawalFee(uint256 newFee) public onlyOwner {
        require(newFee <= MAX_WITHDRAWAL_FEE, ">MaxFee");
        withdrawalFee = newFee;
        emit UpdateWithdrawalFee(withdrawalFee);
    }

    function updateMinWithdrawal(uint256 newMin) public onlyOwner {
        require(newMin > 0, "ZeroMinWithdrawal");
        minWithdrawal = newMin;
        emit UpdateMinWithdrawal(minWithdrawal);
    }
```

Minter.sol #2114-2114

```
    function processWithdrawals(uint256[] calldata withdrawalIds) public onlyOwner {
```

Minter.sol #2134-2134

```
    function collectWithdrawalFees(address receiver) public onlyOwner {
```

Minter.sol #2167-2167

```
    function withdraw(address receiver) public virtual onlyOwner override {
```

Minter.sol #2213-2213

```
    function withdraw(address receiver) public virtual onlyOwner override {
```

Minter.sol #1859-1867

```
    function transferStakingTokenOwnership(address newOwner) public onlyOwner {//@audit CR @audit two
        stakingToken.transferOwnership(newOwner);
        emit TransferStakingTokenOwnership(newOwner);
    }

    function mint(uint256 amount, address receiver) public onlyOwner {
        stakingToken.mint(receiver, amount);
        emit Mint(address(msg.sender), receiver, amount);
    }
```

Minter.sol #1888-1893

```
    function withdraw(address receiver) public virtual onlyOwner {
        uint256 availableBalance = address(this).balance;
        require(availableBalance > 0, "ZeroWithdraw");
        SafeTransferLib.safeTransferETH(receiver, availableBalance);
        emit Withdraw(address(msg.sender), receiver, availableBalance);
    }
```

Minter.sol #1923-1928

```
    function withdraw(address receiver) public virtual onlyOwner {
        uint256 availableBalance = baseToken.balanceOf(address(this));
        require(availableBalance > 0, "ZeroWithdraw");
        baseToken.safeTransferFrom(address(this), receiver, availableBalance);
        emit Withdraw(address(msg.sender), receiver, availableBalance);
    }
```

Recommendation

Consider implementing a decentralized governance mechanism or a multi-signature scheme that requires consensus among multiple parties before pausing or unpausing the contract. This can help mitigate the centralization risk associated with a single owner controlling critical contract functions. Alternatively, you can provide a clear justification for the centralization aspect and ensure that users are aware of the potential risks associated with a single point of control.

Alleviation Acknowledged

The Accumulated Finance replied as below:
In standard design tokens for staking are withdrawn by multi-sig admin and staked using Accumulated Finance Staking Manager automated system that prevents human errors.
If the network provides a permissionless way to stake tokens on behalf of the smart contract, withdrawal by multi-sig admin can be disabled in the contract and additional methods to manage stake on behalf of Minter contract can be implemented.
In standard design LST rewards are minted by multi-sig admin. With the release of AEVM (Accumulated EVM), we plan to mitigate this centralization risk by validating staking rewards, issuing LST rewards, and processing withdrawals through a set of network validators.

Low risk (1)

1. The `minWithdrawal` lacks the upper boundry

Low risk | Security Analyzer

The `minWithdrawal` lacks the upper boundry, as a result, if it is set as the max value of the type uint256, the `requestWithdrawal` function will malfunction, due to the requre check will always fail.

```
function requestWithdrawal(uint256 amount, address receiver) public nonReentrant {
    require(amount >= minWithdrawal, "LessThanMin");
```

File(s) Affected

Minter.sol #2071-2075

```
    function updateMinWithdrawal(uint256 newMin) public onlyOwner {
        require(newMin > 0, "ZeroMinWithdrawal");
        minWithdrawal = newMin;
        emit UpdateMinWithdrawal(minWithdrawal);
    }
```

Recommendation

Adding upper boundry for the `minWithdrawal`.

Alleviation Fixed

This finding is addressed by adding a upper boundry, in commit 03e161dd28c036505e16902904e7d4ef627b45c3.

Informational (4)

1. Unused Return Value

Informational | Security Analyzer

Either the return value of an external call is not stored in a local or state variable, or the return value is declared but never used in the function body.

File(s) Affected

Minter.sol #1908-1908

```
        baseToken.approve(address(this), type(uint256).max);
```

Minter.sol #2136-2136

```
2136        stakingToken.approve(address(this), totalWithdrawalFees);
```

Recommendation

Ensure the return value of external function calls is used.

Alleviation Fixed

The finding is addressed by removing the functions, in commits afc847b6bb695ded7e93806c7e6bb6cab0ad72c9, and fd16775131dabe48fb1bbd68785c6ecaee31ba1a.

2. SafeMath Can be Remove on the Solidity Version 0.8.0 or Above 0.8.0

Informational Security Analyzer

Solidity version is 0.8.0 or above 0.8.0 implemented the overflow checks same as the SafeMath did.

Reference: Arithmetic operations revert on underflow and overflow

File(s) Affected

Minter.sol #2-2

```
2 pragma solidity ^0.8.20;
```

Recommendation

Removing the usage of the SafeMath library.

Alleviation Fixed

This finding is addressed by removing the usage of the SafeMath, in the commit 311ba2576115d64a6b55543204ff99ccf6909d47.

3. The Redundant Check on `request.claimed` From the `processWithdrawals` Function

Informational Security Analyzer

The `processWithdrawals` function will move a withdrawal request's status `processed` from `false` to `true`, and each withdrawal request can only be processed once due to there is a `require` check on the `request.processed`:

```
    function processWithdrawals(uint256[] calldata withdrawalIds) public onlyOwner {
        uint256 totalWithdrawals;
        for (uint256 i = 0; i < withdrawalIds.length; i++) {
            uint256 withdrawalId = withdrawalIds[i];
            WithdrawalRequest storage request = _withdrawalRequests[withdrawalId];
            require(request.amount > 0, "ZeroAmount");
            require(!request.processed, "AlreadyProcessed");
            require(!request.claimed, "AlreadyClaimed");//@audit redundant check
```

Meanwhile, the `request.claimed` status does not be updated in the `processWithdrawals` function. there is no need to check the `request.claimed` status from the `processWithdrawals` function.

File(s) Affected

Minter.sol #2121-2121

```
2121            require(!request.claimed, "AlreadyClaimed");
```

Recommendation

Removing the redundant check on the `request.claimed` status.

Alleviation Fixed

This finding is addressed by removing the redundant check, in the commit 9fae5f3ad8dacee6a6b9c114d26ce512972f71a5.

4. No Need to Use `safeTransferFrom`

Informational Security Analyzer

When deploying the `ERC20Minter` contract, the contract first approves itself a max allowance for the token `baseToken`, in order to call the `safeTransferFrom` function with enough allowance from the `deposit` function and the `withdraw` function.

```
constructor(address _baseToken, address _stakingToken) BaseMinter(_stakingToken) {
    baseToken = IERC20(_baseToken);
    // this contract can spend baseToken
    baseToken.approve(address(this), type(uint256).max);
}
...
function withdraw(address receiver) public virtual onlyOwner {
    uint256 availableBalance = baseToken.balanceOf(address(this));
    require(availableBalance > 0, "ZeroWithdraw");
    baseToken.safeTransferFrom(address(this), receiver, availableBalance);
    emit Withdraw(address(msg.sender), receiver, availableBalance);
}
```

However, there is no need to do the `approve + safeTransferFrom` actions if the from address in the `safeTransferFrom` function is the contract itself, it can be totally replace them with one `safeTransfer` function.

The similar case happens on the below functions:

- The `redeem` function of the `ERC20MinterRedeem` contract.
- The `collectWithdrawalFees` function of the `BaseMinterWithdrawal` contract.
- Functions `withdraw`, and `claimWithdrawal` of the `ERC20MinterWithdrawal` contract.

File(s) Affected

Minter.sol #1908-1908

```
1908        baseToken.approve(address(this), type(uint256).max);
```

Minter.sol #1918-1918

```
1918        baseToken.safeTransferFrom(address(msg.sender), address(this), amount);
```

Minter.sol #1926-1926

```
1926        baseToken.safeTransferFrom(address(this), receiver, availableBalance);
```

Minter.sol #2134-2140

```
2134    function collectWithdrawalFees(address receiver) public onlyOwner {
2135        require(totalWithdrawalFees > 0, "ZeroFees");
2136        stakingToken.approve(address(this), totalWithdrawalFees);
2137        stakingToken.safeTransferFrom(address(this), receiver, totalWithdrawalFees);
2138        totalWithdrawalFees = 0;
2139        emit CollectWithdrawalFees(address(msg.sender), receiver, totalWithdrawalFees);
2140    }
```

Minter.sol #2228-2228

```
2228        baseToken.safeTransferFrom(address(this), receiver, request.amount);
```

Minter.sol #1991-1991

```
1991        baseToken.safeTransferFrom(address(this), receiver, redeemAmount);
```

Minter.sol #2216-2216

```
2216        baseToken.safeTransferFrom(address(this), receiver, balance);
```

Recommendation

Replacing the **`approve + safeTransferFrom`** actions with the **`safeTransfer`** when the from address in the **`safeTransferFrom`** function is the contract itself.

Alleviation Fixed

This finding is addressed by replacing the `approve + safeTransferFrom` actions with the `safeTransfer`, in commits fd16775131dabe48fb1bbd68785c6ecaee31ba1a and afc847b6bb695ded7e93806c7e6bb6cab0ad72c9.

Disclaimer

This report is governed by the stipulations (including but not limited to service descriptions, confidentiality, disclaimers, and liability limitations) outlined in the Services Agreement, or as detailed in the scope of services and terms provided to you, the Customer or Company, within the context of the Agreement. The Company is permitted to use this report only as allowed under the terms of the Agreement. Without explicit written permission from MetaTrust, this report must not be shared, disclosed, referenced, or depended upon by any third parties, nor should copies be distributed to anyone other than the Company.

It is important to clarify that this report neither endorses nor disapproves any specific project or team. It should not be viewed as a reflection of the economic value or potential of any product or asset developed by teams or projects engaging MetaTrust for security evaluations. This report does not guarantee that the technology assessed is completely free of bugs, nor does it comment on the business practices, models, or legal compliance of the technology's creators.

This report is not intended to serve as investment advice or a tool for investment decisions related to any project. It represents a thorough assessment process aimed at enhancing code quality and mitigating risks inherent in cryptographic tokens and blockchain technology. Blockchain and cryptographic assets inherently carry ongoing risks. MetaTrust's role is to support companies and individuals in their security diligence and to reduce risks associated with the use of emerging and evolving technologies. However, MetaTrust does not guarantee the security or functionality of the technologies it evaluates.

MetaTrust's assessment services are contingent on various dependencies and are continuously evolving. Accessing or using these services, including reports and materials, is at your own risk, on an as-is and as-available basis. Cryptographic tokens are novel technologies with inherent technical risks and uncertainties. The assessment reports may contain inaccuracies, such as false positives or negatives, and unpredictable outcomes. The services may rely on multiple third-party layers.

All services, labels, assessment reports, work products, and other materials, or any results from their use, are provided "as is" and "as available," with all faults and defects, without any warranty. MetaTrust expressly disclaims all warranties, whether express, implied, statutory, or otherwise, including but not limited to warranties of merchantability, fitness for a particular purpose, title, non-infringement, and any warranties arising from course of dealing, usage, or trade practice. MetaTrust does not guarantee that the services, reports, or materials will meet specific requirements, be error-free, or be compatible with other software, systems, or services.

Neither MetaTrust nor its agents make any representations or warranties regarding the accuracy, reliability, or currency of any content provided through the services. MetaTrust is not liable for any content inaccuracies, personal injuries, property damages, or any loss resulting from the use of the services, reports, or materials.

Third-party materials are provided "as is," and any warranty concerning them is strictly between the Customer and the third-party owner or distributor. The services, reports, and materials are intended solely for the Customer and should not be relied upon by others or shared without MetaTrust's consent. No third party or representative thereof shall have any rights or claims against MetaTrust regarding these services, reports, or materials.

The provisions and warranties of MetaTrust in this agreement are exclusively for the Customer's benefit. No third party has any rights or claims against MetaTrust regarding these provisions or warranties. For clarity, the services, including any assessment reports or materials, should not be used as financial, tax, legal, regulatory, or other forms of advice.