Guard

Smart contracts security assessment

Final report

Tariff: Standard

Champion Finance ChamETF

October 2022

0xguard.com

hello@0xguard.com

Contents

1. Introduction 3
2. Contracts checked 4
3. Procedure 4
4. Known vulnerabilities checked 4
5. Classification of issue severity 5
6. Issues 6
7. Conclusion 10
8. Disclaimer 11
9. Slither output 12

Introduction

The report has been prepared for **Champion Finance ChamETF**.

Contract ChamETF is an ERC20-like token. Users can mint (burn) this token by providing (withdrawing) other pool tokens. There is a fee of up to 1% for burning.

The ChamETF token can be sold (or bought), but the contract owner (operator) can set a fee of up to 10% for selling (buying).

The ChamETF contract use upgradeable deployment scheme. Users have to trust the owner, who can change the contract's logic at their will.

The code is available at the GitHub repository and was audited after the commit 4cec5b81f141c090a46ccecf6ee9604b30f9f4ad.

Report Update.

The contract's code was updated according to this report and rechecked after the commit 25b4aaab84723420760127a46d954275802a2c91.

Only 2 contracts were audited: ChamETF, ERC20WithTaxUpgradeable.

The contract is deployed through a proxy contract 0x6197f1b4198296b637b731E9994BC366d29cCcaa on Avalanche C-chain. The current and audited implementation is located at 0xAF921f4145BdF146DAe177F59C1A1Bfe67D6B61E.

Name	Champion Finance ChamETF
Audit date	2022-10-26 - 2022-10-28
Language	Solidity
Platform	Avalanche Network

Contracts checked

Name	Address
ERC20WithTaxUpgradeable	0xAF921f4145BdF146DAe177F59C1A1Bfe67D6B61E
ChamETF	0xAF921f4145BdF146DAe177F59C1A1Bfe67D6B61E

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed

Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Blocking and loss of tokens (ChamETF)

Status: Fixed

The contract operator can remove tokens from the pool using the `removeTokenAsset()` function. This can lead to the following problems.

1. Following such action, the deposited tokens will be locked on the contract. Moreover, among the locked ones, there may be user tokens that were added using the `joinPool()` function. Thus, users will be able to leave the pool (`exitPool()`) only **with a loss**.

2. Losses are possible in the following case. The user could join the pool when it has 3 tokens, for example, WBTC.e, USDC.e, WETH.e. By investing 100 of each token. After that, the operator will remove one of the tokens, for example, WBTC.e. If the user decides to leave the pool, he will receive back only 100 USDC.e and 100 WETH.e. (commissions are ignored).

3. In case the operator deletes all assets, users will never be able to burn ChamETF tokens using the `exitPool()` function. At least one token must be left.

Recommendation: Consider restricting the operator's rights to delete assets when users have

already joined the pool. Otherwise, the contract architecture needs to be fine-tuned to prevent the loss of users when assets are removed.

Medium severity issues

1. Multiple call of function initialize() (ChamETF)

Status: Fixed

There is a way to call the `initialize()` function multiple times. To do this, the operator must remove all previously added tokens using the `removeTokenAsset()` function. After that, a repeated call of the `initialize()` function with other tokens will be available. This operation can be repeated an unlimited number of times.

```
function initialize(
        address[] memory _tokens,
        uint256[] memory balances,
        address tokenProvider
    ) public onlyOperator {
        require(tokens.length == 0, "ChamETF: Already initialized");
        ...
    }
```

Recommendation: Add a flag state variable to define the initialization.

Low severity issues

1. Gas optimization (ERC20WithTaxUpgradeable)

Status: Fixed

Visibility of the functions `name()`, `symbol()`, `decimals()`, `balanceOf()`, `transfer()`, `approve()`, `transferFrom()`, `increaseAllowance()`, `decreaseAllowance()`, `setMarketLpPairs()` can be declared as '`external`' to save gas.

2. Lack of events (ChamETF)
Status: Fixed

We recommend adding events for all 'setter' functions.

3. Token duplication (ChamETF)
Status: Fixed

The same token can be used multiple times in the `_tokens` parameter of the `initialize()` function.

This will cause the token to be transferred multiple times, but the balance (`Record.balance`) will only be written down on the last transfer.

Recommendation: Add validation to prevent repeating tokens in the `initialize()` function.

4. Using re-entracy guard (ChamETF)
Status: Fixed

Since any token can be added by the operator, we recommend using the `nonReentrant()` modifier for all functions where tokens are transferred. This will prevent unwanted actions when using unknown tokens.

5. Gas optimization (ChamETF)
Status: Fixed

1. Visibility of the functions `initialize()`, `addTokenAsset()`, `removeTokenAsset()`, `setMinBoundTokens()`, `setMaxBoundTokens()`, `setExitFee()`, `setMaxPoolTokens()`, `setExitFeeRecipient()`, `setMinimumBalance()`, `joinPool()`, `exitPool()`, `getUsedBalance()`, `getCurrentTokens()` can be declared as '`external`' to save gas.

2. There may be situations where the fees (`_exitFee`) may be zero when calling the `exitPool()` function. Consider adding a non-zero check to prevent zero transfers.

```
function exitPool(uint256 poolAmountIn, uint256[] memory minAmountsOut) public {
  ...
  uint256 _exitFee = bmul(poolAmountIn, exitFee);
  ...
```

```
    _transfer(caller, exitFeeRecipient, _exitFee);
    ...
}
```

3. Writing to memory type `memory` (`record.ready = true`) is useless on L301 because this value will not be stored in the blockchain:

```
    function _updateInputToken(
        address token,
        Record memory record,
        uint256 realBalance
    ) internal {
        if (!record.ready && realBalance >= record.balance) {
            minimumBalances[token] = 0;
            records[token].ready = true;
            record.ready = true;
            emit TokenReady(token);
        }
        records[token].balance = realBalance;
    }
```

Conclusion

Champion Finance ChamETF ERC20WithTaxUpgradeable, ChamETF contracts were audited. 1 high, 1 medium, 5 low severity issues were found.
1 high, 1 medium, 5 low severity issues have been fixed in the update.

We recommend writing tests to cover the founded issues.

The contract owner (operator) can set a fee of up to 10% for selling (buying) tokens. At the same time, the contract is upgradeable, it is going to be deployed via proxies. The owner can make changes to the contract implementation (including the calculation of taxes). Users interacting with the contract have to trust the owner.

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 0xGuard 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 0xGuard 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.
OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

Slither output

```
ChamETF.__ChamETF__init(string,string,address)._name (etf/ChamETF.sol#43) shadows:
        - ERC20WithTaxUpgradeable._name (etf/ERC20WithTaxUpgradeable.sol#44) (state
variable)
ChamETF.__ChamETF__init(string,string,address)._symbol (etf/ChamETF.sol#44) shadows:
        - ERC20WithTaxUpgradeable._symbol (etf/ERC20WithTaxUpgradeable.sol#45) (state
variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-
shadowing

ChamETF.setExitFee(uint256) (etf/ChamETF.sol#142-145) should emit an event for:
        - exitFee = _exitFee (etf/ChamETF.sol#144)
ChamETF.setMaxPoolTokens(uint256) (etf/ChamETF.sol#147-149) should emit an event for:
        - maxPoolTokens = _maxPoolTokens (etf/ChamETF.sol#148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-
arithmetic

ERC20WithTaxUpgradeable.setTaxWallet(address)._wallet (etf/
ERC20WithTaxUpgradeable.sol#80) lacks a zero-check on :
                - taxWallet = _wallet (etf/ERC20WithTaxUpgradeable.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation

ChamETF._pullUnderlying(address,address,uint256) (etf/ChamETF.sol#260-277) has external
calls inside a loop: (success,data) = erc20.call(abi.encodeWithSelector(IERC20.transferF
rom.selector,from,address(this),amount)) (etf/ChamETF.sol#265-272)
ChamETF._pushUnderlying(address,address,uint256) (etf/ChamETF.sol#279-291) has external
calls inside a loop: (success,data) =
erc20.call(abi.encodeWithSelector(IERC20.transfer.selector,to,amount)) (etf/
ChamETF.sol#284-286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-
a-loop

Reentrancy in ChamETF.addTokenAsset(address,uint256,uint256) (etf/ChamETF.sol#87-106):
        External calls:
        - _pullUnderlying(token,msg.sender,balance) (etf/ChamETF.sol#103)
                - (success,data) = erc20.call(abi.encodeWithSelector(IERC20.transferFrom
.selector,from,address(this),amount)) (etf/ChamETF.sol#265-272)
```

```
        State variables written after the call(s):
        - minimumBalances[token] = minimumBalance (etf/ChamETF.sol#104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-2

Reentrancy in ChamETF.addTokenAsset(address,uint256,uint256) (etf/ChamETF.sol#87-106):
        External calls:
        - _pullUnderlying(token,msg.sender,balance) (etf/ChamETF.sol#103)
                - (success,data) = erc20.call(abi.encodeWithSelector(IERC20.transferFrom
.selector,from,address(this),amount)) (etf/ChamETF.sol#265-272)
        Event emitted after the call(s):
        - TokenAdded(token,minimumBalance) (etf/ChamETF.sol#105)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-3

Pragma version^0.8.0 (etf/ERC20WithTaxUpgradeable.sol#4) allows old versions
Pragma version0.8.16 (etf/ChamETF.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity

Low level call in ChamETF._pullUnderlying(address,address,uint256) (etf/
ChamETF.sol#260-277):
        - (success,data) = erc20.call(abi.encodeWithSelector(IERC20.transferFrom.selecto
r,from,address(this),amount)) (etf/ChamETF.sol#265-272)
Low level call in ChamETF._pushUnderlying(address,address,uint256) (etf/
ChamETF.sol#279-291):
        - (success,data) =
erc20.call(abi.encodeWithSelector(IERC20.transfer.selector,to,amount)) (etf/
ChamETF.sol#284-286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls

Function ERC20WithTaxUpgradeable.__ERC20_init(string,string) (etf/
ERC20WithTaxUpgradeable.sol#63-65) is not in mixedCase
Function ERC20WithTaxUpgradeable.__ERC20_init_unchained(string,string) (etf/
ERC20WithTaxUpgradeable.sol#67-72) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.setTaxRate(uint256)._value (etf/
ERC20WithTaxUpgradeable.sol#75) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.setTaxWallet(address)._wallet (etf/
```

```
ERC20WithTaxUpgradeable.sol#80) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.setMarketLpPairs(address,bool)._pair (etf/
ERC20WithTaxUpgradeable.sol#85) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.setMarketLpPairs(address,bool)._value (etf/
ERC20WithTaxUpgradeable.sol#85) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.excludeTaxAddress(address)._address (etf/
ERC20WithTaxUpgradeable.sol#89) is not in mixedCase
Parameter ERC20WithTaxUpgradeable.includeTaxAddress(address)._address (etf/
ERC20WithTaxUpgradeable.sol#93) is not in mixedCase
Variable ERC20WithTaxUpgradeable.__gap (etf/ERC20WithTaxUpgradeable.sol#457) is not in
mixedCase
Variable ERC20WithTaxUpgradeable.TAX_MULTIPLIER (etf/ERC20WithTaxUpgradeable.sol#52) is
not in mixedCase
Function ChamETF.__ChamETF__init(string,string,address) (etf/ChamETF.sol#42-58) is not
in mixedCase
Parameter ChamETF.__ChamETF__init(string,string,address)._name (etf/ChamETF.sol#43) is
not in mixedCase
Parameter ChamETF.__ChamETF__init(string,string,address)._symbol (etf/ChamETF.sol#44)
is not in mixedCase
Parameter ChamETF.__ChamETF__init(string,string,address)._exitFeeRecipient (etf/
ChamETF.sol#45) is not in mixedCase
Parameter ChamETF.initialize(address[],uint256[],address)._tokens (etf/ChamETF.sol#62)
is not in mixedCase
Parameter ChamETF.setMinBoundTokens(uint256)._minBoundTokens (etf/ChamETF.sol#126) is
not in mixedCase
Parameter ChamETF.setMaxBoundTokens(uint256)._maxBoundTokens (etf/ChamETF.sol#134) is
not in mixedCase
Parameter ChamETF.setExitFee(uint256)._exitFee (etf/ChamETF.sol#142) is not in
mixedCase
Parameter ChamETF.setMaxPoolTokens(uint256)._maxPoolTokens (etf/ChamETF.sol#147) is not
in mixedCase
Parameter ChamETF.setExitFeeRecipient(address)._exitFeeRecipient (etf/ChamETF.sol#151)
is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions

ERC20WithTaxUpgradeable.__ERC20_init_unchained(string,string) (etf/
ERC20WithTaxUpgradeable.sol#67-72) uses literals with too many digits:
        - TAX_MULTIPLIER = 100000 (etf/ERC20WithTaxUpgradeable.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits
```

```
ERC20WithTaxUpgradeable.__gap (etf/ERC20WithTaxUpgradeable.sol#457) is never used in
ChamETF (etf/ChamETF.sol#12-319)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-
variable

setMarketLpPairs(address,bool) should be declared external:
        - ERC20WithTaxUpgradeable.setMarketLpPairs(address,bool) (etf/
ERC20WithTaxUpgradeable.sol#85-87)
name() should be declared external:
        - ERC20WithTaxUpgradeable.name() (etf/ERC20WithTaxUpgradeable.sol#100-102)
symbol() should be declared external:
        - ERC20WithTaxUpgradeable.symbol() (etf/ERC20WithTaxUpgradeable.sol#108-110)
decimals() should be declared external:
        - ERC20WithTaxUpgradeable.decimals() (etf/ERC20WithTaxUpgradeable.sol#125-127)
balanceOf(address) should be declared external:
        - ERC20WithTaxUpgradeable.balanceOf(address) (etf/
ERC20WithTaxUpgradeable.sol#139-141)
transfer(address,uint256) should be declared external:
        - ERC20WithTaxUpgradeable.transfer(address,uint256) (etf/
ERC20WithTaxUpgradeable.sol#151-155)
approve(address,uint256) should be declared external:
        - ERC20WithTaxUpgradeable.approve(address,uint256) (etf/
ERC20WithTaxUpgradeable.sol#174-178)
transferFrom(address,address,uint256) should be declared external:
        - ERC20WithTaxUpgradeable.transferFrom(address,address,uint256) (etf/
ERC20WithTaxUpgradeable.sol#196-205)
increaseAllowance(address,uint256) should be declared external:
        - ERC20WithTaxUpgradeable.increaseAllowance(address,uint256) (etf/
ERC20WithTaxUpgradeable.sol#219-223)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20WithTaxUpgradeable.decreaseAllowance(address,uint256) (etf/
ERC20WithTaxUpgradeable.sol#239-248)
initialize(address[],uint256[],address) should be declared external:
        - ChamETF.initialize(address[],uint256[],address) (etf/ChamETF.sol#61-85)
addTokenAsset(address,uint256,uint256) should be declared external:
        - ChamETF.addTokenAsset(address,uint256,uint256) (etf/ChamETF.sol#87-106)
removeTokenAsset(address) should be declared external:
        - ChamETF.removeTokenAsset(address) (etf/ChamETF.sol#108-124)
setMinBoundTokens(uint256) should be declared external:
        - ChamETF.setMinBoundTokens(uint256) (etf/ChamETF.sol#126-132)
```

```
setMaxBoundTokens(uint256) should be declared external:
        - ChamETF.setMaxBoundTokens(uint256) (etf/ChamETF.sol#134-140)
setExitFee(uint256) should be declared external:
        - ChamETF.setExitFee(uint256) (etf/ChamETF.sol#142-145)
setMaxPoolTokens(uint256) should be declared external:
        - ChamETF.setMaxPoolTokens(uint256) (etf/ChamETF.sol#147-149)
setExitFeeRecipient(address) should be declared external:
        - ChamETF.setExitFeeRecipient(address) (etf/ChamETF.sol#151-160)
setMinimumBalance(address,uint256) should be declared external:
        - ChamETF.setMinimumBalance(address,uint256) (etf/ChamETF.sol#162-170)
joinPool(uint256,uint256[]) should be declared external:
        - ChamETF.joinPool(uint256,uint256[]) (etf/ChamETF.sol#173-204)
exitPool(uint256,uint256[]) should be declared external:
        - ChamETF.exitPool(uint256,uint256[]) (etf/ChamETF.sol#206-244)
getUsedBalance(address) should be declared external:
        - ChamETF.getUsedBalance(address) (etf/ChamETF.sol#246-253)
getCurrentTokens() should be declared external:
        - ChamETF.getCurrentTokens() (etf/ChamETF.sol#255-257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
```

0x Guard