CHAINPROOF

Audit Report

Apollo Caps ETF

March 2024

Network BSC

Address 0xcB7bF0218CCBf340C6676706C60A41c1E9CBdD44

Audited by © ChainProof

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MEE	Missing Events Emission	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	ACE
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://bscscan.com/address/0xcb7bf0218ccbf340c6676706c60a41c1e9cbdd44
Address	0xcb7bf0218ccbf340c6676706c60a41c1e9cbdd44
Network	BSC
Symbol	ACE
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

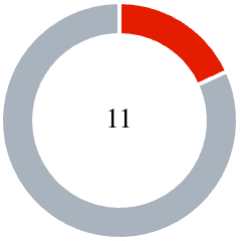
Filename	SHA256
ACE.sol	7344f0939364d55c29e51fdd51f62d97c2ccf35f1f0decfd20dce217933b6d87

Audit Updates

Initial Audit	25 Mar 2024

Source Files

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/ACE.sol#L36
Status	Unresolved

Description

The contract owner has the authority to pause or unpause the transactions for all users including the owner.

```
function pause() public onlyOwner {
    _pause();
} function unpause() public
onlyOwner {
    _unpause();
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/ACE.sol#L133
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling `addBlackList` thefunction.

```
function addBlackList(address account) public onlyOwner {
 _blacklist[account] = true;
       emit Blacklist(account,
                      true);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/ACE.sol#L45,143
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setTaxRate(
    uint16 _buyTaxRate,
    uint16 _sellTaxRate
) public onlyOwner {
    require(
  _buyTaxRate <=
 _maxBuyTaxRate,
   "ACE: buy tax
    rate must be
    less than or
    equal to max
buy tax rate"
    );
   require(
  _sellTaxRate <=
 _maxSellTaxRate,
   "ACE: sell tax
     rate must be
     less than or
     equal to max
sell tax rate"
    );
  buyTaxRate =
  _buyTaxRate;
 sellTaxRate =
 _sellTaxRate;
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	ACE.sol#L95
Status	Unresolved

Description

The contract sends fees to the `receiveFeeAddress` as part of the transfer flow. If this address is set to zero then it will revert since the ERC20 library reverts in case of zero address. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (taxAmount > 0) {
    _transfer(sender, receiveFeeAddress, taxAmount);
}
```

Recommendation

The team is advised to either prevent the `receiveFeeAddress` from being configured with zero value or not take fees when the address is zero.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ACE.sol#L45,71,75,143
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setTaxRate(
    uint16 _buyTaxRate,
    uint16 _sellTaxRate
) public onlyOwner {
    require(
  _buyTaxRate <=
 _maxBuyTaxRate,
   "ACE: buy tax
    rate must be
    less than or
    equal to max
buy tax rate"
    );
   require(
  _sellTaxRate <=
 _maxSellTaxRate,
   "ACE: sell tax
     rate must be
     less than or
     equal to max
sell tax rate"
    );
  buyTaxRate =
  _buyTaxRate;
 sellTaxRate =
 _sellTaxRate;
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ACE.sol#L12,14
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint16 private _maxBuyTaxRate = 1000
uint16 private _maxSellTaxRate = 1000
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

-

L04 Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ACE.sol#L45,46,145,150
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint16 _buyTaxRate uint16
_sellTaxRate address
_receiveFeeAddress
```

```
address _to address
_token
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with. Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

-

L07 Missing Events Arithmetic

Criticality	Minor / Informative
Location	ACE.sol#L56
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
buyTaxRate = _buyTaxRate
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

-

L16 Validate Variable Setters

Criticality	Minor / Informative
Location	ACE.sol#L146
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
receiveFeeAddress = _receiveFeeAddress
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 Stable Compiler Version

Criticality	Minor / Informative

-

Location	ACE.sol#L3
Status	Unresolved

Description

`^` Thesymbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.20;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

-

L20 Succeeded Transfer Check

Criticality	Minor / Informative
Location	ACE.sol#L154
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(_to, balance)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-

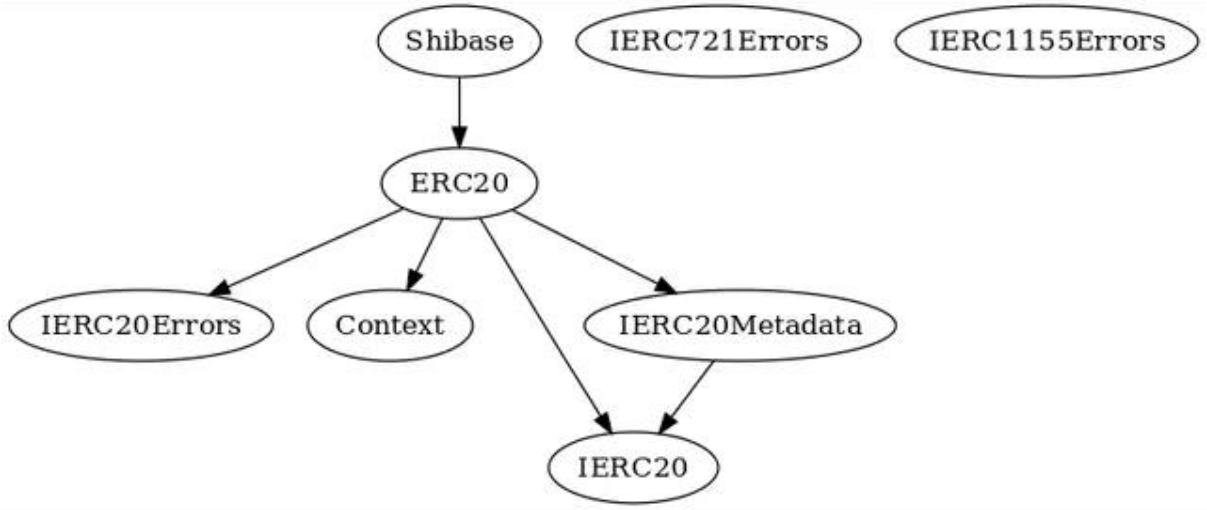
	approve	External	✓	-
	transferFrom	External	✓	-

IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-

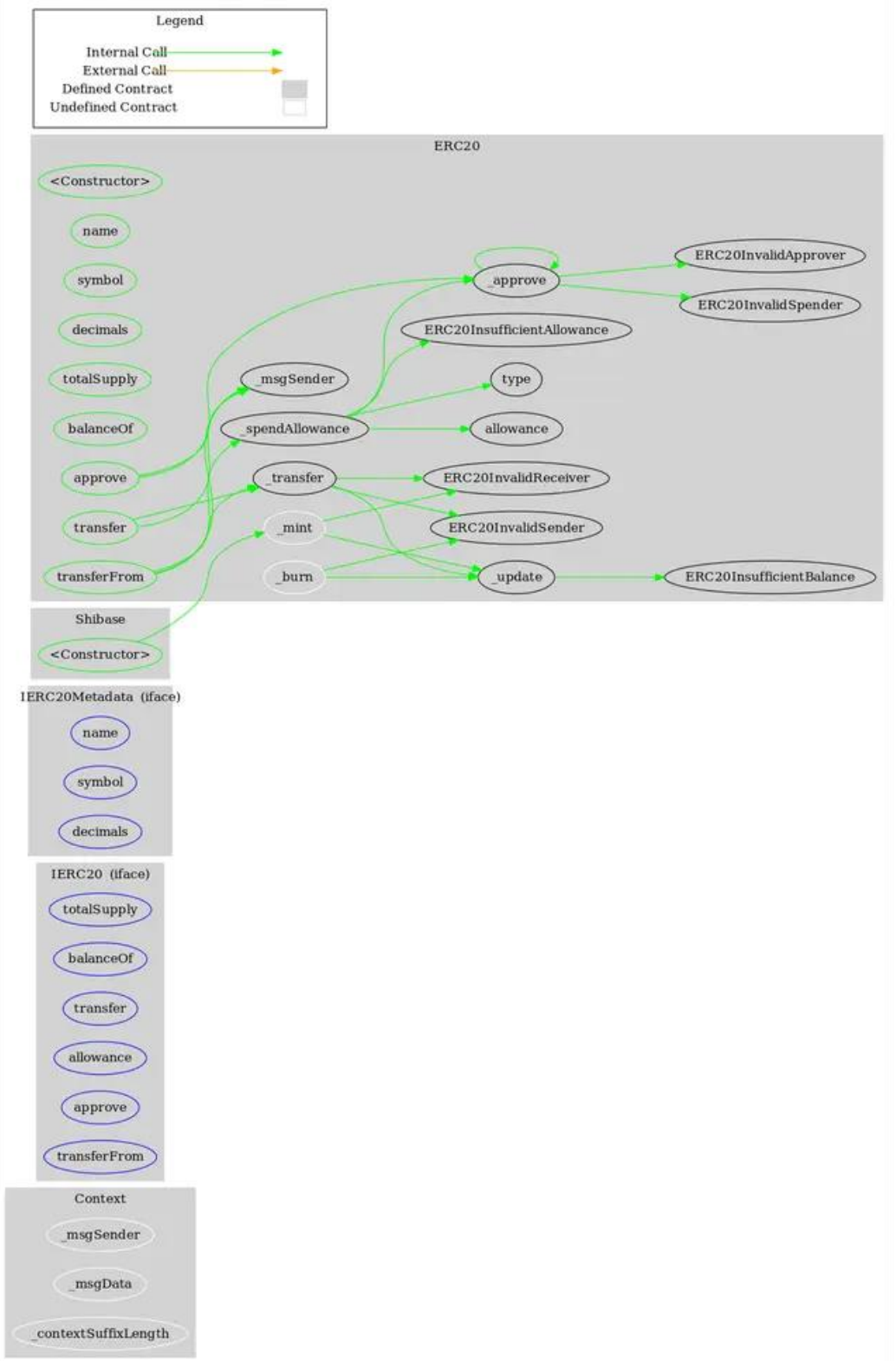
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
Shibase	Implementation	ERC20		
		Public	✓	ERC20

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
_approve
ERC20InvalidApprover
ERC20InvalidSpender
ERC20InsufficientAllowance
_msgSender
type
_spendAllowance
allowance
_transfer
ERC20InvalidReceiver
_mint
ERC20InvalidSender
_burn
_update
ERC20InsufficientBalance
Shibase
<Constructor>
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
_contextSuffixLength

Summary

Apollo Caps ETF contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Chainproof’s 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 regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Chainproof to perform a security assessment. This document 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 represents an extensive assessment 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk Chainproof’s position is that each company and individual are responsible for their own due diligence and continuous security Chainproof’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Chainproof are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The

assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About ChainProof

Chainproof is an Audit & KYC firm for Blockchain Projects, aimed at securing the Blockchain and the assets at risk. Chainproof is fueled by Industry grade experienced Blockchain Developers from all around the globe. From finding vulnerabilities, potential scams, malicious code mitigation, improper implementation of the token which can lead to loss of user's fund, you name it and we cover and secure them all.

Security testing and risk mitigation is given the highest priority at ChainProof. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.
ChainProof is aiming to make crypto discoverable and efficient globally. We associate with extremely robust testing and code review, leaving no room for any security risks because, when it comes to user's funds, we need to leave no stone unturned. Cheers!

The ChainProof team

ChainProof.dev