Cyberscope

Audit Report

Aimeme Presale

January 2024

Network	ETH
Address	0x4089dFC5C51883003B57412c3EA6e92c436Bac5C
Audited by	© cyberscope

Table of Contents

Review

Explorer	https://etherscan.io/address/0x4089dfc5c51883003b57412c3ea6e92c436bac5c

Audit Updates

Initial Audit	09 Jan 2024

Source Files

Filename	SHA256
contracts/Presale.sol	8f646f7616ddc123fd2d5eb9663c5c184052aae7ae8e8f8f2546015e9e927efd
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/Address.sol	1e0922f6c0bf6b1b8b4d480dcabb691b1359195a297bde6dc5172e79f3a1f826
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	fa36a21bd954262006d806b988e4495562e7b50420775e2aa0deecb596fd1902
@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol	3e7aa0e0f69eec8f097ad664d525e7b3f0a3fda8dcdd97de5433ddb131db86ef
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231

Overview

The "Presale" contract, is tailored for managing a presale event of the “Aim” token. Key features of this contract include its integration with a pricing aggregator for real-time price feeds, support for Ethereum (ETH) and USD-based stablecoin (USDT) transactions, and multi-phase sale dynamics. The contract is structured to offer a nuanced control over the presale phases, token pricing, and sales caps, ensuring an adaptable and responsive presale environment.

Initialization

Upon deployment, the contract initializes essential parameters such as token quantities for sale, price tiers, and the USD threshold for purchases. It also sets critical addresses, including the burn wallet, the token (AIM), and the USDT address. The contract establishes a connection with an aggregator interface for live price data.

Token Pricing and Sales Phases

The presale is segmented into five stages, each with designated token quantities and dynamic pricing. The token prices are set in a tiered array, responding to time-based triggers. This structure facilitates a flexible pricing mechanism that adapts to different stages of the presale.

Token Purchasing

The contract supports token purchases using ETH and USDT. For ETH transactions, it converts ETH to USD value based on the latest exchange rate from the pricing aggregator. In the case of USDT, it checks the user's USDT balance and allowance, ensuring compliance with the sale's terms. The contract meticulously calculates the token amount corresponding to the paid USD, adhering to the current pricing tier.

Stage Management and Presale Status

The contract's logic includes functions to transition through different sale stages, updating the current stage based on the elapsed time and tokens sold. It also allows the owner to

start the presale, and it automatically toggles the presale and claim statuses based on predefined conditions.

Claim Functionality

Post-presale, users can claim their purchased tokens. The contract ensures claims are only made post-presale and validates the claim amounts against the user's purchase record.

Administrative Controls

The contract empowers the owner with several administrative functions, including the ability to initiate the presale, withdraw accumulated ETH or USDT, and manage private sale allocations. It also features a burn mechanism for unsold tokens at each stage's end.

Security and Compliance

Throughout, the contract employs checks to prevent common vulnerabilities, such as reentrancy attacks, and adheres to standard practices like SafeERC20 for token transfers. It also ensures compliance with sale terms, like purchase limits and sale duration.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ITU	Incomplete Token Update	Unresolved
●	REA	Redundant Event Argument	Unresolved
●	TUU	Time Units Usage	Unresolved
●	CR	Code Repetition	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

ITU - Incomplete Token Burn Process

Criticality	Minor / Informative
Location	contracts/Presale.sol#L185
Status	Unresolved

Description

Function `updateStage()` transfers the remaining tokens of the current stage to a burn wallet, symbolizing the burning of these tokens. The function correctly calculates the number of tokens to burn `(tokenToBurn)` based on `remainingToSell[currentStage]` and transfers these tokens to the `burnWallet`. However, after this transfer, the `remainingToSell[currentStage]` is not updated to reflect that these tokens have been burned. As a result, the contract's state does not accurately represent the actual remaining tokens, since the tokens that have been sent to the burn wallet are still counted as part of `remainingToSell`.

```
uint256 tokenToBurn = remainingToSell[currentStage];
    if(tokenToBurn > 0)
    {
        IERC20(AIM).safeTransfer(address(burnWallet),
tokenToBurn);
    }
```

Recommendation

It is recommended to modify this function's logic, in order to accurately update the `remainingToSell[currentStage]` variable. This change will ensure the contract's state correctly reflects the reduced token supply following the burn and maintain the integrity of the presale process, ensuring the contract's data accurately represents the actual token distribution and supply.

REA - Redundant Event Argument

Criticality	Minor / Informative
Location	contracts/Presale.sol#L201
Status	Unresolved

Description

Inside the `startPresale` function, the `PresaleStatusUpdated` event with `true` argument is raised. This event is designed to indicate a change in the presale status. However, the use of the true argument in the event is redundant, as the startPresale function is already structured to activate the presale, thereby implicitly setting the presale status to true. This redundancy in the event argument is unnecessary because the event PresaleStatusUpdated is only emitted in scenarios where the presale status is set to active, making the explicit passing of true as an argument redundant.

```
function startPresale() external onlyOwner{
    require(!presaleStatus && !claimStatus, "Presale is already
start");
    require(IERC20(AIM).balanceOf(address(this)) >= 150000000 *
10**18, "Token not available to start presale");

    startTime = block.timestamp;
    presaleStatus = true;
    emit PresaleStatusUpdated(true);
}
```

Recommendation

Consider removing the status argument from the PresaleStatusUpdated event. Since the startPresale function's sole purpose is to set the presale status to active, the emission of the event itself is sufficient to indicate this change. This change will simplify the event and align it with the principle of minimizing unnecessary code and complexity in smart contracts.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/Presale.sol#L71,83,191
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 currentRound = (block.timestamp - startTime) / 604800;

uint256 round = (block.timestamp - startTime) / 604800;

startTime += 2419200;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/Presale.sol#L71,112
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
uint256 currentRound = (block.timestamp - startTime) / 604800;
if(currentRound > 3)
{
    _updateStage();
}
if(currentStage > 4)
{
    presaleStatus = false;
    claimStatus = true;
}
else
{
    uint256 round = (block.timestamp - startTime) / 604800;
            round = round > 3 ? 3 : round;
    uint256 price = tokenPrice[currentStage][round];
...
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	contracts/Presale.sol#L30,37
Status	Unresolved

Description

Variable `tokenToSell` is declared and initialized with specific values but remains unused throughout the entire contract. No functions or methods within the contract utilize this variable, indicating that it serves no practical purpose. Additionally, variable `clamedToken` is redundant, since `claimToken` function can only be called once by the user. Redundancy in variable declaration, therefore, not only contributes to unnecessary consumption of storage and computational resources but also adds to the complexity and potential confusion in understanding the contract's functionality.

```
uint256[5]  public tokenToSell;

struct buyTokenInfo {
    uint256 USDPaid;
    uint256 tokenFromBuy;
    uint256 clamedToken;
}
```

Recommendation

It is recommended that these variables be removed from the contract. This removal will streamline the contract, reducing the consumption of gas and storage resources.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/Presale.sol#L56,59,60,61
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
USDThreshold
burnWallet
AIM
USDT
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Presale.sol#L17,20,23,24,34
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.

```
uint256 public USDRaised
uint256 public USDThreshold
address public AIM
address public USDT

struct buyTokenInfo {
      uint256 USDPaid;
      uint256 tokenFromBuy;
      uint256 clamedToken;
    }
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/Presale.sol#L107,129
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 amount = (getLatestPrice() * msg.value) / (10**30)
uint256 requestedToken = amount * (10**18) / price
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

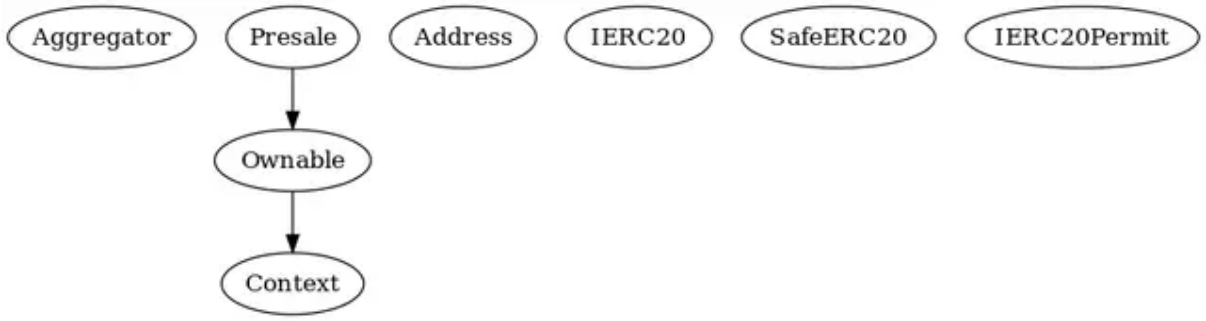
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Aggregator	Interface			
	latestRoundData	External		-
Presale	Implementation	Ownable		
		Public	✓	-
	buyWithUSDT	External	✓	-
	buyWithETH	External	Payable	-
	privateSaleBuyers	External	✓	onlyOwner
	_buytokens	Internal	✓	
	_updateStage	Internal	✓	
	startPresale	External	✓	onlyOwner
	claimToken	External	✓	-
	withdrawETH	Public	✓	onlyOwner
	withdrawUSDT	Public	✓	onlyOwner
	getLatestPrice	Public		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

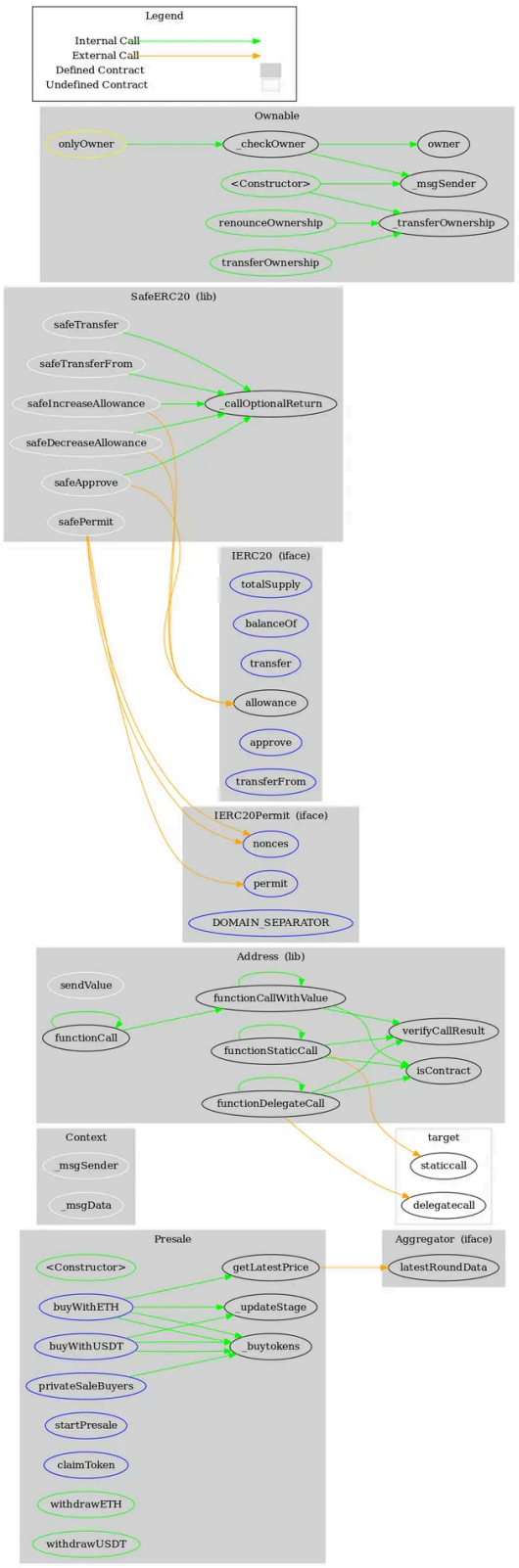
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			

	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeIncreaseAllowance
_callOptionalReturn
safeDecreaseAllowance
safeApprove
safePermit
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IERC20Permit (iface)
nonces
permit
DOMAIN_SEPARATOR
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
Context
_msgSender
_msgData
target
staticcall
delegatecall
Presale
<Constructor>
getLatestPrice
buyWithETH
_updateStage
buyWithUSDT
_buytokens
privateSaleBuyers
startPresale
claimToken
withdrawETH
withdrawUSDT
Aggregator (iface)
latestRoundData

Summary

Aimeme’s presale contract implements a presale mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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 Cyberscope'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 Cyberscope 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope 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 Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io