Cyberscope

Audit Report

PAWDNAH

June 2023

SHA256 9f04fd49cca58a4849c42726b70b44ac3fe452442f5de4e8bc00278459453 3b5

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x24ba719f4e996f225dda1e1a27db16a560b51a4c

Audit Updates

Initial Audit	06 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/7-burn2/v1/audit.pdf
Corrected Phase 2	14 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/7-burn2/v2/audit.pdf
Corrected Phase 3	23 Jun 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/AccessControl.sol	afd98330d27bddff0db7cb8fcf42bd4766dda5f60b40871a3bec6220f9c9edf7
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/security/Pausable.sol	2072248d2f79e661c149fd6a6593a8a3f038466557c9b75e50e0b001bcb5cf97
@openzeppelin/contracts/security/ReentrancyGuard.sol	fa97ea556c990ee44f2ef4c80d4ef7d0af3f5f9b33a02142911140688106f5a9
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	b7383c48331f3cc9901fc05e5d5830fcd533699a77f3ee1e756a98681bfbb2ee
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	c8309ed2c1c7edf52a23833798c94507702248debfc4ed1f645e571e3c230f8b
@openzeppelin/contracts/utils/Address.sol	8b85a2463eda119c2f42c34fa3d942b61aee65df381f48ed436fe8edb3a7d602
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/Math.sol	85a2caf3bd06579fb55236398c1321e15fd524a8fe140dff748c0f73d7a52345

@openzeppelin/contracts/utils/math/SignedMath.sol	420a5a5d8d94611a04b39d6cf5f0249255 2ed4257ea82aba3c765b1ad52f77f6
@openzeppelin/contracts/utils/Strings.sol	cb2df477077a5963ab50a52768cb74ec6f3 2177177a78611ddbbe2c07e2d36de
contracts/SecureDeposit.sol	9f04fd49cca58a4849c42726b70b44ac3fe 452442f5de4e8bc002784594533b5

Overview

The SecureDeposit contract implements a rewards mechanism based on deposits. The users can deposit a specific amount of tokens. The deposits are tracked from the contract using a FIFO (First in First out) structure. If the total number of depositors is more than three and the contract has three times the tokens of the first depositor, then the first depositor is applicable to withdraw three times the deposited amount and dequeued from the structure. The tokens are intended to be USDC. The deposited amount is defined by the contract owner. The depositors cannot withdraw their deposits unless they are applicable.

Roles

Users

- deposit
- withdraw

Admin

- pause
- unpause
- onlyRole
- addFundsToBackupWallet
- setDepositAmount
- revertState

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	4	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	RNRM	Redundant No Reentrant Modifier	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

RNRM - Redundant No Reentrant Modifier

Criticality	Minor / Informative
Location	contracts/testingDeploy/SecureDeposit.sol#L120
Status	Unresolved

Description

The contract uses the `nonReentrant` modifier to the `withdraw()` method, which suggests an intention to prevent potential reentrancy attacks. However, the `withdraw()` method exclusively deals with the `usdc` token, which is considered a trusted source within the contract.

Given that the `usdc` token is a trusted entity and no external address can be executed through the `withdraw()` method, the risk of reentrancy vulnerabilities is effectively mitigated. Consequently, the usage of the `nonReentrant` modifier becomes redundant, adding unnecessary complexity to the codebase.

```
function withdraw(uint256 amount) public whenNotPaused nonReentrant
{
    require(eligibleWithdrawals[msg.sender] >= amount,
"Insufficient eligible withdrawal balance.");

    // Transfer the requested amount to the depositor
    usdc.safeTransfer(msg.sender, amount);

    // Update the withdrawal eligibility amount
    eligibleWithdrawals[msg.sender] -= amount;

    emit Withdrawn(msg.sender, amount);
}
```

Recommendation

To address this finding and enhance code simplicity and clarity, it is recommended to remove the unnecessary "nonReentrant" modifier from the "withdraw()" method. By removing the modifier, the code becomes more streamlined and easier to comprehend, reducing the gas consumption.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/SecureDeposit.sol#L73,74
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.

```
withdrawalWalletBalance
totalDeposits
```

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/SecureDeposit.sol#L152
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 _newAmount
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/SecureDeposit.sol#L3
Status	Unresolved

Description

The `^` symbol 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 ^0.8.19;
```

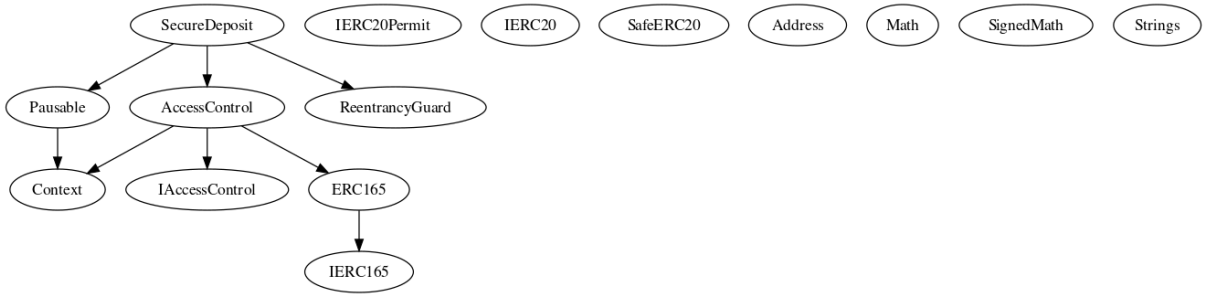
Recommendation

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.

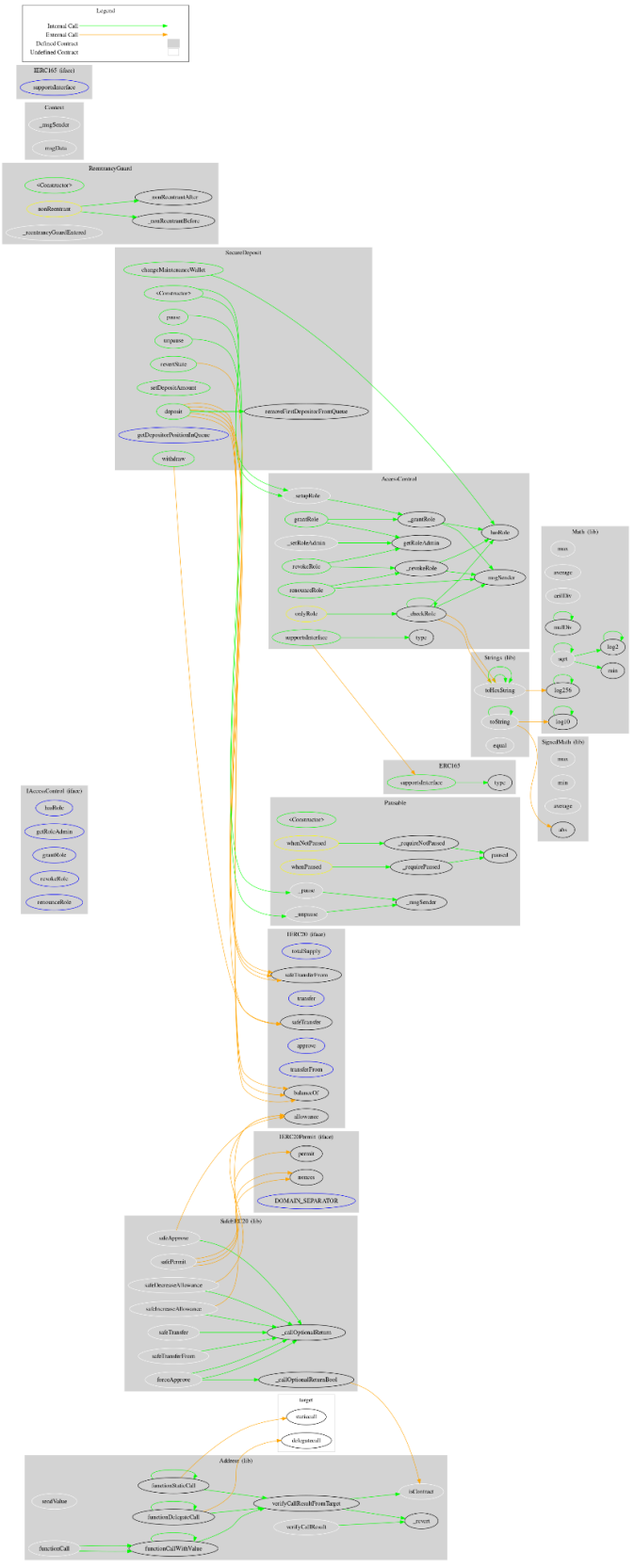
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SecureDeposit	Implementation	AccessControl, ReentrancyGuard, Pausable		
		Public	✓	-
	deposit	Public	✓	whenNotPaused
	withdraw	Public	✓	whenNotPaused nonReentrant
	removeFirstDepositorFromQueue	Internal	✓	
	pause	Public	✓	onlyRole
	unpause	Public	✓	onlyRole
	setDepositAmount	Public	✓	onlyRole
	changeMaintenanceWallet	Public	✓	-
	getDepositorPositionInQueue	External		-
	revertState	Public	✓	onlyRole

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC165 (Iface)
supportsInterface
Context
_msgSender
_msgData
ReentrancyGuard
<Constructor>
nonReentrant
nonReentrantAfter
nonReentrantBefore
_reentrancyGuardEntered
SecureDeposit
changeMaintenanceWallet
<Constructor>
pause
unpause
revertState
setDepositAmount
deposit
removeFirstDepositFromQueue
getDepositedPositionInQueue
withdraw
AccessControl
setupRole
grantRole
_setRoleAdmin
revokeRole
renounceRole
only Role
supportsInterface
_grantRole
getRoleAdmin
_revokeRole
_checkRole
hasRole
msgSender
type
Math (Lib)
max
average
ceilDiv
mulDiv
sqrt
log2
min
log256
log10
Strings (Lib)
toHexString
toString
equal
SignedMath (Lib)
max
min
average
abs
ERC165
supportsInterface
type
IAccessControl (Iface)
hasRole
getRoleAdmin
grantRole
revokeRole
renounceRole
Pausable
<Constructor>
whenNotPaused
whenPaused
_pause
_unpause
_requireNotPaused
_requirePaused
paused
msgSender
IERC20 (Iface)
totalSupply
safeTransferFrom
transfer
safeTransfer
approve
transferFrom
balanceOf
allowance
IERC20Permit (Iface)
permit
nonces
DOMAIN_SEPARATOR
SafeERC20 (Lib)
safeApprove
safePermit
safeDecreaseAllowance
safeIncreaseAllowance
safeTransfer
safeTransferFrom
forceApprove
_callOptionalReturn
_callOptionalReturnBool
target
staticcall
delegatecall
Address (Lib)
sendValue
functionStaticCall
functionDelegateCall
functionCall
functionCallWithValue
verifyCallResultFromTarget
verifyCallResult
isContract
revert

Summary

PAWDNAH contract implements a rewards mechanism based on deposits. 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