Audit Report

GoldBrick

February 2024

Network	BSC
Address	0x6605a6d03C2238C20F21ADF2bA5D206bAf3FDB86
Audited by	© cyberscope

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	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ROA	Redundant Ownership Assignment	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	UMFO	Unnecessary Mint Function Override	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	GBCKToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6605a6d03c2238c20f21adf2ba5d206baf3fdb86
Address	0x6605a6d03c2238c20f21adf2ba5d206baf3fdb86
Network	BSC
Symbol	GBCK
Decimals	18
Total Supply	15,000,000

Audit Updates

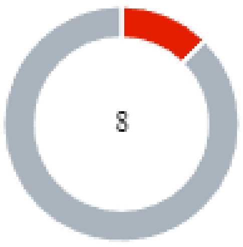
Initial Audit	06 Feb 2024

Source Files

Filename	SHA256
contracts/GBCKToken.sol	899b07168aa53147cb71c9127381453820d686698f8cfd864d4590bc59dbc10a
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d

@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef8c4e410ae99608be5964d98fa701
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	0c8a43f12ac2081c6194d54da96f02ebc457760d6514f6b940689719fcef8c0a
@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol	3e7aa0e0f69eec8f097ad664d525e7b3f0a3fda8dcdd97de5433ddb131db86ef
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231

Findings Breakdown



● Critical		1
● Medium		0
● Minor / Informative		7

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/GBCKToken.sol#L77
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users, excluding the addresses that are in the `allowedTransfer` mapping, which is managed by the contract owner. The owner may take advantage of it by setting the `publicSaleStartDate` to a very high value.

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual override {
    require(
        block.timestamp >= publicSaleStartDate ||
allowedTransfer[sender],
        "GBCKToken: transfer not allowed"
    );
    super._transfer(sender, recipient, amount);
}

function setPublicSaleStartDate(
    uint256 _publicSaleStartDate
) public onlyOwner {
    require(
        _publicSaleStartDate > block.timestamp,
        "GBCKToken: invalid date"
    );
    publicSaleStartDate = _publicSaleStartDate;
}
```

Recommendation

The contract could embody a check for not allowing setting the `publicSaleStartDate` to a very high value. 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.

ROA - Redundant Ownership Assignment

Criticality	Minor / Informative
Location	contracts/GBCKToken.sol#L20
Status	Unresolved

Description

The contract inherits from OpenZeppelin's `Ownable` contract, which by design automatically assigns the contract's deployer `msg.sender` as the owner upon deployment. However, within the constructor, there is an explicit call to `_transferOwnership(msg.sender);`, which redundantly sets the ownership to the deployer again. This additional operation does not alter the contract's behavior or ownership status, as the ownership assignment is inherently handled by the `Ownable` contract's constructor. The presence of this redundant operation increases the deployment gas cost slightly and introduces unnecessary complexity to the contract code.

```
constructor(
    string memory _name,
    string memory _symbol,
    address _tokensReceiver
) ERC20(_name, _symbol) {
    _transferOwnership(msg.sender);
    uint256 extraTokens = 500_000 * 10 ** 18;
    allowedTransfer[_tokensReceiver] = true;
    allowedTransfer[msg.sender] = true;
    _mint(_tokensReceiver, extraTokens);
    _mint(msg.sender, _cap - extraTokens);
}
```

Recommendation

It is recommended to remove the `_transferOwnership(msg.sender);` call from the constructor. This change will streamline the contract by eliminating unnecessary code, thereby reducing the gas cost for contract deployment and enhancing code readability. Simplifying the constructor in this manner aligns with best practices for smart contract development, focusing on efficiency and clarity. After the removal, the contract's ownership will still be correctly assigned to the deployer `msg.sender`, as guaranteed by the inherited `Ownable` contract, ensuring that the intended functionality and security posture is maintained.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/GBCKToken.sol#L58
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 setPublicSaleStartDate(
    uint256 _publicSaleStartDate
) public onlyOwner {
    require(
        _publicSaleStartDate > block.timestamp,
        "GBCKToken: invalid date"
    );
    publicSaleStartDate = _publicSaleStartDate;
}
```

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.

UMFO - Unnecessary Mint Function Override

Criticality	Minor / Informative
Location	contracts/GBCKToken.sol#L38
Status	Unresolved

Description

The contract, which is designed to issue a fixed supply of tokens at deployment, utilizes the overridden `_mint` function solely within its constructor to allocate tokens to specific addresses. The original intent behind the override appears to ensure that token minting operations do not exceed the predefined cap. However, given that all minting operations are encapsulated within the constructor and calculated to respect the supply cap, the additional complexity introduced by overriding the `_mint` function for the purpose of enforcing the cap is deemed unnecessary. This complexity does not contribute additional security or functionality under the contract's operational parameters, where the total supply is fixed and minting is confined to the initial deployment phase.

```
function _mint(address account, uint256 amount) internal
virtual override {
    require(
        ERC20.totalSupply() + amount <= cap(),
        "GBCKToken: cap exceeded"
    );
    super._mint(account, amount);
}
```

Recommendation

It is recommended to streamline the contract by removing the override of the `_mint` function that includes the supply cap check. This simplification would not impact the contract's functionality or security, as the supply cap is inherently respected through the controlled minting operations within the constructor. Removing the override would reduce the contract's complexity and gas costs associated with deployment, leading to a more efficient and straightforward implementation.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/GBCKToken.sol#L50,59
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.

```
address _address
uint256 _publicSaleStartDate
```

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	contracts/GBCKToken.sol#L65
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.

```
publicSaleStartDate = _publicSaleStartDate
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/GBCKToken.sol#L16,17
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/GBCKToken.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;
```

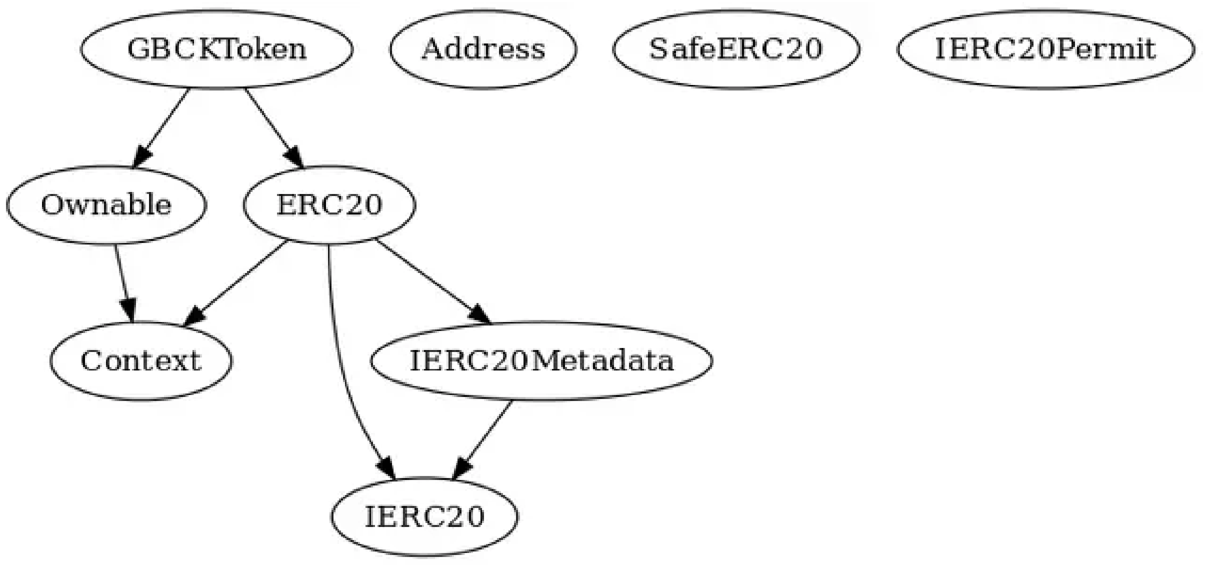
Recommendation

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.

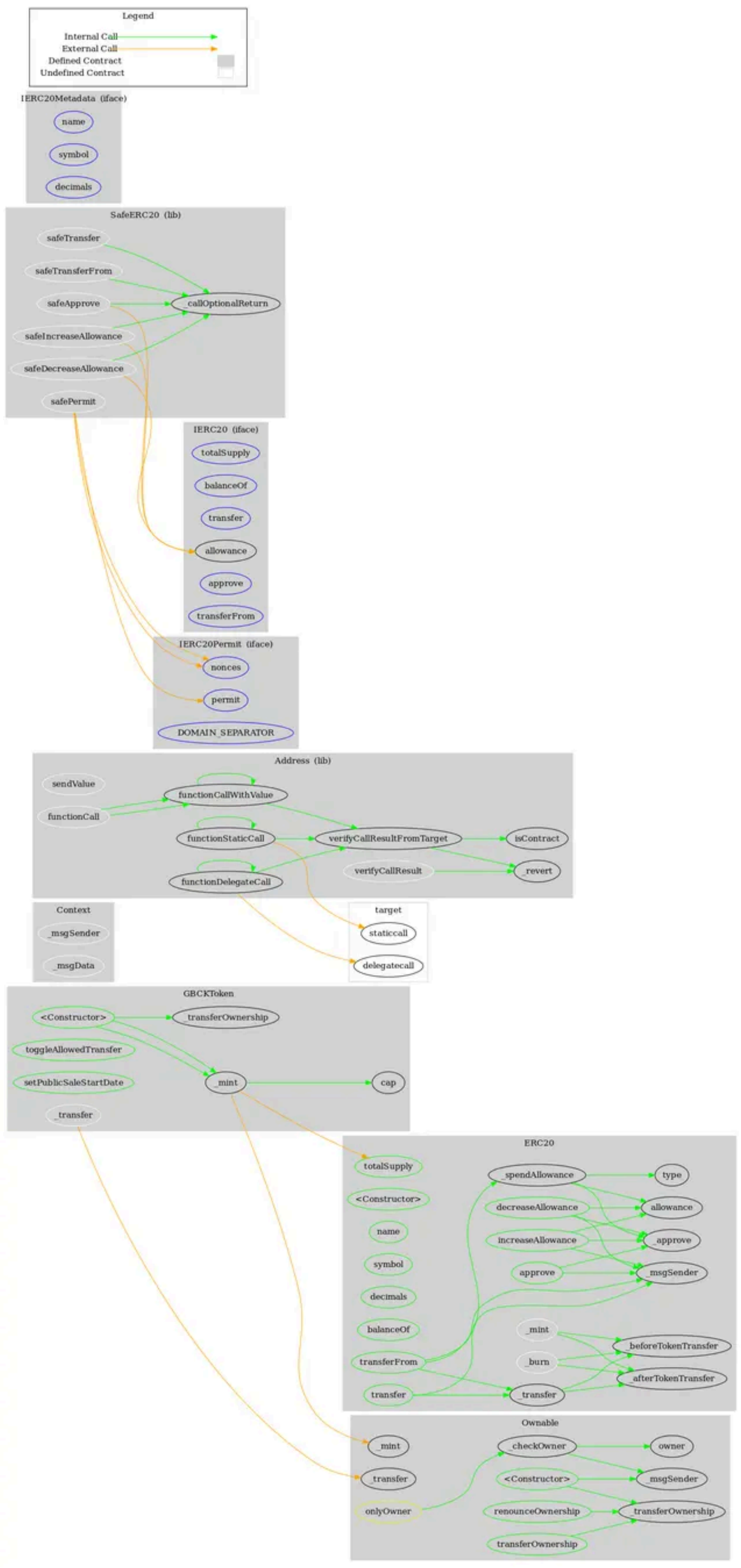
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
GBCKToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	cap	Public		-
	_mint	Internal	✓	
	toggleAllowedTransfer	Public	✓	onlyOwner
	setPublicSaleStartDate	Public	✓	onlyOwner
	_transfer	Internal	✓	

Inheritance Graph


$$\begin{tikzcd}
 & GBCKToken \arrow[dl] \arrow[d] & Address & SafeERC20 & IERC20Permit \\
Ownable \arrow[d] & ERC20 \arrow[dl] \arrow[dr] \arrow[dd, bend right] & & & \\
Context & & IERC20Metadata \arrow[dl] & & \\
 & IERC20 & & &
\end{tikzcd}$$

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
_callOptionalReturn
safeIncreaseAllowance
safeDecreaseAllowance
safePermit
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IERC20Permit (iface)
nonces
permit
DOMAIN_SEPARATOR
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
verifyCallResult
_revert
functionDelegateCall
Context
_msgSender
_msgData
target
staticcall
delegatecall
GBCKToken
<Constructor>
transferOwnership
toggleAllowedTransfer
setPublicSaleStartDate
_mint
cap
_transfer
ERC20
totalSupply
_spendAllowance
type
<Constructor>
decreaseAllowance
allowance
name
increaseAllowance
_approve
symbol
approve
_msgSender
decimals
balanceOf
_mint
_beforeTokenTransfer
transferFrom
_burn
_afterTokenTransfer
transfer
_transfer
Ownable
_mint
_checkOwner
owner
_transfer
<Constructor>
_msgSender
onlyOwner
renounceOwnership
_transferOwnership
transferOwnership

Summary

GoldBrick 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. 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

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