Cyberscope

Audit Report

ShadowGold

April 2024

SHA256 30e6e7789fdf8c1e517e2b2a497f249f424ead703d5e9e713ec38973f345a35d

SHA256 7081988eb4acf884fee741e43239a619a0aac9fb9bb7a5baf084012c61a92bd0

Audited by © cyberscope

Table of Contents

Review

SDG.sol	https://testnet.bscscan.com/address/0xed3BE6dfcd9563e34dE1528c98414753cCd9799A
DividendDistributor.sol	https://testnet.bscscan.com/address/0x6f7d10fF58cb3FD15Fc7c4a67c5f2a8c6A7143fd

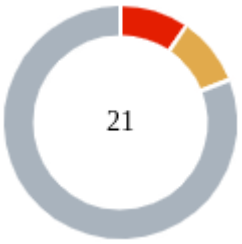
Audit Updates

Initial Audit	03 Apr 2024 https://github.com/cyberscope-io/audits/blob/main/shadowfi/v1/audit.pdf
Corrected Phase 2	15 Apr 2024 https://github.com/cyberscope-io/audits/blob/main/shadowfi/v2/audit.pdf
Corrected Phase 3	28 Apr 2024

Source Files

Filename	SHA256
SDG.sol	30e6e7789fdf8c1e517e2b2a497f249f424ead703d5e9e713ec38973f345a35d
SDGDistributor.sol	7081988eb4acf884fee741e43239a619a0aac9fb9bb7a5baf084012c61a92bd0

Findings Breakdown



Critical	2
Medium	2
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	2	0	0
Medium	0	2	0	0
Minor / Informative	0	17	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	ST	Stops Transactions	Acknowledged
	US	Untrusted Source	Acknowledged
	BC	Blacklists Addresses	Acknowledged
	UDB	Unupdated Distributor Balances	Acknowledged
	DDP	Decimal Division Precision	Acknowledged
	IDI	Immutable Declaration Improvement	Acknowledged
	ISC	Ineffective Supply Check	Acknowledged
	MEM	Misleading Error Messages	Acknowledged
	MMN	Misleading Method Naming	Acknowledged
	MIV	Missing Index Verification	Acknowledged
	PAV	Pair Address Validation	Acknowledged
	PLPI	Potential Liquidity Provision Inadequacy	Acknowledged
	PVC	Price Volatility Concern	Acknowledged
	RFV	Redundant Fee Variable	Acknowledged

●	RRS	Redundant Require Statement	Acknowledged
●	RSML	Redundant SafeMath Library	Acknowledged
●	OCTD	Transfers Contract's Tokens	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L07	Missing Events Arithmetic	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged
●	L20	Succeeded Transfer Check	Acknowledged

ST - Stops Transactions

Criticality	Critical
Location	SDG.sol#L630
Status	Acknowledged

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (!allowedAddresses[msg.sender] && !allowedAddresses[recipient]) {
    require(
        block.timestamp > transferBlockTime,
        "Transfers have not been enabled yet."
    );
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in section `US` . As a result, the contract might operate as a honeypot.

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Team Update

The team has acknowledged that this is not a security issue and states:

This function is necessary to be able to launch smoothly. We will need to airdrop our existing holders their allotment from their holding of the previous ShadowFi(SDF) token on Binance Smart Chain. This function has a safety measure to ensure it cannot be utilized in perpetuity.

Once the lock time has passed and transfers are enabled globally, the lock time cannot be updated again to stop transactions going forward.

US - Untrusted Source

Criticality	Critical
Location	SDG.sol#L801
Status	Acknowledged

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract that can be set as the `distributor` address is untrusted. As a result, it may produce security issues and harm the transactions.

```
    function setDistributorAndFeeReceiver(
        address _distributorContract,
        address _feeReceiver
    ) public onlyOwner {
        distributor = IDividendDistributor(_distributorContract);
        isDividendExempt[_distributorContract] = true;
        isFeeExempt[_feeReceiver] = true;
    }
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

Team Update

The team has acknowledged that this is not a security issue and states:

The contract owner will have the ability to replace the distributor contract as new features are added into the ecosystem. This is a necessary mechanism that allows the ecosystem to adapt and grow into the future.

BC - Blacklists Addresses

Criticality	Medium
Location	SDG.sol#L636
Status	Acknowledged

Description

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

```
require(
    !blackList[sender] && !blackList[recipient],
    "Either the spender or recipient is blacklisted."
);
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

The contract owner will have the ability to blacklist specific wallet addresses from participating in the ShadowGold ecosystem. This function is primarily needed to prevent unwanted exchange listings. The ShadowGold ecosystem relies on all trading to take place within the trading pairs that are supported by the ecosystem. This ensures fees are generated for every exchange or swap of the token, which is a critical aspect to the ShadowGold ecosystem.

UDB - Unupdated Distributor Balances

Criticality	Medium
Location	SDG.sol#L640
Status	Acknowledged

Description

The contract contains the `_transferFrom` which immediately executes the `_basicTransfer` function in case the `if` condition condition is met, resulting in a transfer transaction before updating the balances of the distributor. Consequently, the updated balances will not be reflected accurately in the distribution contract, leading to discrepancies in token distribution.

```
    function _transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) internal returns (bool) {
    ...

    if (IDividendDistributor(distributor).checkInSwap()) {
        return _basicTransfer(sender, recipient, amount);
    }
    ...
```

Recommendation

It is recommended to ensure that balances of the distributor are updated before executing any transfer transactions. Implement mechanisms to update the distributor balances synchronously with token transfers to maintain consistency in token distribution.

Team Update

The team has acknowledged that this is not a security issue and states:

Distributor itself never has shares. It is exempted. The only time checkInSwap is true, is if the distributor is swapping to the LP address. The LP address is also exempted from shares.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	DividendDistributor.sol#L919
Status	Acknowledged

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 swapAmount = swapThreshold.mul(marketingFee).div(totalBuyFee);
...
uint256 amountSDGReflection = swapThreshold.mul(reflectionFee).div(
    totalBuyFee
);
uint256 amountSDGReceiver = swapThreshold.mul(sdgReceiverFee).div(
    totalBuyFee
);
uint256 amountSDGBuyback = swapThreshold.mul(buybackFee).div(
    totalBuyFee
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

Team Update

The team has acknowledged that this is not a security issue and states:

While we understand the importance of maximum decimal precision, we have decided to keep the original decimal precision from previous deployments intact. It has proven for several years of operation to be a non-issue.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	SDG.sol#L518,522,526
Status	Acknowledged

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.

```
maticPair
paxgPair
uniswapUniversalRouter
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

The gas savings only applies for the deployment cost. We are not concerned with deployment cost due to deploying on Polygon and it having incredibly inexpensive gas cost for transactions.

ISC - Ineffective Supply Check

Criticality	Minor / Informative
Location	SDG.sol#L716
Status	Acknowledged

Description

The contract is using a `require` statement in the `processFee` function to ensure that the `_totalSupply` does not exceed `_maxSupply`, with the condition described as a preventive measure against minting new tokens. However, both `_totalSupply` and `_maxSupply` are set to identical values, which inherently means that `_totalSupply` will never exceed `_maxSupply`. This results in the require check always evaluating to true, rendering it redundant and inefficient. This unnecessary check consumes gas and adds complexity without providing any functional benefit or security enhancement.

```
    uint256 constant _totalSupply = 10 ** 8 * (10 ** _decimals);
     uint256 constant _maxSupply = 10 ** 8 * (10 ** _decimals);


     function processFee(address sender, uint256 feeAmount) internal {
         _balances[address(distributor)] =
_balances[address(distributor)].add(
             feeAmount
         );
         require(
             _totalSupply <= _maxSupply,
             "Minting new tokens is not allowed."
         );

         emit Transfer(sender, address(distributor), feeAmount);
     }
```

Recommendation

It is recommended to remove the redundant require check from the `processFee` function, as the conditions checked are statically defined and will never trigger the provided error condition. Removing this check will reduce gas costs for transactions involving this

function and simplify the contract’s codebase. If future adjustments to supply mechanics are anticipated, consider implementing dynamic checks that reflect actual operational conditions. Otherwise, ensure that static conditions are evaluated and streamlined during the initial contract review to avoid similar inefficiencies.

Team Update

The team has acknowledged that this is not a security issue and states:

The gas savings to remove this check are very minimal. This check may be necessary to insure future additions to the ecosystem can never accidentally cause a over-mint.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	SDG.sol#L716
Status	Acknowledged

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(
    _totalSupply <= _maxSupply,
    "Minting new tokens is not allowed."
);
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

Team Update

The team has acknowledged that this is not a security issue and states:

The error message will not be misleading if and when that requirement check is ever utilized.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	SDG.sol#L856
Status	Acknowledged

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

Specifically, the `getCirculatingSupply` function calculate the circulating supply but behaves similarly to `getMaxCirculatingSupply` , since this function subtracts the balance of a "DEAD" address and a "ZERO" address from the maximum supply, potentially providing misleading information about the actual circulating supply.

```
    function getCirculatingSupply() public view returns (uint256) {
        return _maxSupply.sub(balanceOf(DEAD)).sub(balanceOf(ZERO));
    }
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code. It is recommended to revise the `getCirculatingSupply` function to accurately reflect the circulating supply by excluding addresses that are not actively participating in the circulation from the total supply.

Team Update

The team has acknowledged that this is not a security issue and states:

This contract has gone through several developers over the years of it's operation. Each developer has utilized different naming conventions. No edit to address this will be made as it poses no functional threat to the contract.

MIV - Missing Index Verification

Criticality	Minor / Informative
Location	SDG.sol#L272,286,300
Status	Acknowledged

Description

The contract contains the `authorizeForMultiplePermissions`, `unauthorizeFor`, and `unauthorizeForMultiplePermissions` functions, all relying on `permIndex` for permission handling. However, these functions lack verification to ensure the existence of the `permIndex` before its usage. Consequently, users could define a `permIndex` that does not exist, leading to potential unauthorized access or unintended behavior.

```
    function authorizeForMultiplePermissions(
        address adr,
        string[] calldata permissionNames
    ) public authorizedFor(Permission.Authorize) {
        for (uint256 i; i < permissionNames.length; i++) {
            uint256 permIndex =
permissionNameToIndex[permissionNames[i]];
            authorizations[adr][permIndex] = true;
            emit AuthorizedFor(adr, permissionNames[i], permIndex);
        }
    }

    function unauthorizeFor(
        address adr,
        string memory permissionName
    ) public authorizedFor(Permission.Unauthorize) {
        require(adr != owner, "Can't unauthorize owner");

        uint256 permIndex = permissionNameToIndex[permissionName];
        authorizations[adr][permIndex] = false;
        emit UnauthorizedFor(adr, permissionName, permIndex);
    }

    function unauthorizeForMultiplePermissions(
        address adr,
        string[] calldata permissionNames
    ) public authorizedFor(Permission.Unauthorize) {
        require(adr != owner, "Can't unauthorize owner");

        for (uint256 i; i < permissionNames.length; i++) {
            uint256 permIndex =
permissionNameToIndex[permissionNames[i]];
            authorizations[adr][permIndex] = false;
            emit UnauthorizedFor(adr, permissionNames[i], permIndex);
        }
    }
```

Recommendation

It is recommended to enhance the functions by incorporating additional checks to validate the existence of the `permIndex` before executing operations. This would mitigate the risk of unauthorized access and ensure the contract behaves as intended.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this, the ShadowAuth authorization library has been utilized within this contract for years with no issues. The contract owner is very familiar with the permission authorization functions.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	DividendDistributor.sol#L633,819,843
Status	Acknowledged

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
address[] memory path = new address[](2);
path[0] = address(_token);
path[1] = address(PAXG);
...
function buyTokensWETH(uint256 amount, address to) internal swapping {
    address[] memory path = new address[](2);
    path[0] = address(WETH);
    path[1] = address(_token);
    ...

function buyTokensPAXG(uint256 amount, address to) internal swapping {
    address[] memory path = new address[](2);
    path[0] = address(PAXG);
    path[1] = address(_token);
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application.

Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	DividendDistributor.sol#L648
Status	Acknowledged

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
address[] memory path = new address[](2);
path[0] = address(_token);
path[1] = address(PAXG)
router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    amount,
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by

executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. The community is well aware of the fact that a project without liquidity has many issues.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DividendDistributor.sol#L907
Status	Acknowledged

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
    function swapBack() public swapping onlyToken {
        uint256 swapAmount =
swapThreshold.mul(marketingFee).div(totalBuyFee);
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. We are aware that the swapThreshold value must be kept at a reasonable value to ensure price impact is not affected greatly.

RFV - Redundant Fee Variable

Criticality	Minor / Informative
Location	DividendDistributor.sol#L922,942
Status	Acknowledged

Description

The contract is utilizing two separate variables, `sdgReceiverFee` and `buybackFee`, to calculate the total amount to be transferred to the `sdgFeeReceiver`. These variables are used to determine portions of the `swapThreshold` that correspond to different fees before being summed up for the final transfer amount. The calculation involves multiplying the `swapThreshold` by each fee and dividing by the `totalBuyFee`, resulting in `amountSDGReceiver` and `amountSDGBuyback` respectively. Subsequently, these amounts are added together for the transfer to sdgFeeReceiver. This approach, while mathematically sound, introduces unnecessary complexity and redundancy since the addition of these variables does not implement any additional functionality or differentiation in the handling of fees. Essentially, the contract is performing an extra step without a clear benefit, which could lead to confusion, increased gas costs, and potential errors in future modifications.

```
uint256 amountSDGReceiver = swapThreshold.mul(sdgReceiverFee).div(
    totalBuyFee
);
uint256 amountSDGBuyback = swapThreshold.mul(buybackFee).div(
    totalBuyFee
);
...
if (amountSDGReceiver > 0 || amountSDGBuyback > 0) {
    try
        IERC20(address(_token)).transfer(
            sdgFeeReceiver,
            amountSDGReceiver + amountSDGBuyback
        )
    {
        emit ReceiverAmount(amountSDGBuyback, amountSDGReceiver);
```

Recommendation

It is recommended to simplify the fee structure by consolidating `sdgReceiverFee` and `buybackFee` into a single variable. This can be achieved by either combining their values into a single fee variable or by re-evaluating the necessity of distinguishing these fees if they ultimately serve a similar purpose and are directed to the same receiver. Simplification will not only reduce the contract's complexity but also minimize potential points of failure and optimize gas costs associated with these calculations and transactions. Additionally, this change would make the contract more straightforward, enhancing its readability and maintainability. Future updates or audits will benefit from a clearer understanding of the fee handling mechanism, thereby reducing the risk of errors or unintended consequences.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. While this appears redundant currently, in the future it will not be.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	SDG.sol#L64 DividendDistributor.sol#L64
Status	Acknowledged

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. Due to being on Polygon, the gas savings from this is negligible.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SDG.sol
Status	Acknowledged

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. We understand that solidity 0.8.0+ has built-in overflow/underflow protection already.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	SDG.sol#L923 DividendDistributor.sol#L969
Status	Acknowledged

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `withdrawTokens` function.

```
   function withdrawTokens(address _token, uint256 _amount) public
onlyOwner {
        IERC20(_token).transfer(owner, _amount);
    }
...
    function withdrawTokens(address token, uint256 _amount) public
onlyOwner {
        IERC20(token).transfer(owner, _amount);
    }
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this. The contract owner has had the ability to withdraw tokens from the contract for many years already.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SDG.sol#L475,476,477,479,480,481,484,485,802,803,851,852,889,900,906,916,917,933,937,962,972,979,980
Status	Acknowledged

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.

```
string constant _name = "ShadowGold"
string constant _symbol = "SDG"
uint8 constant _decimals = 9
uint256 constant _totalSupply = 10 ** 8 * (10 ** _decimals)
uint256 constant _maxSupply = 10 ** 8 * (10 ** _decimals)
uint256 public _maxTxAmount
mapping(address => uint256) _balances
mapping(address => mapping(address => uint256)) _allowances
address _distributorContract
address _feeReceiver
uint256 _minPeriod
uint256 _minDistribution
uint256 GWEI
uint256 _amount

...
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

This contract has gone through several developers over the years of it's operation. Each developer has utilized different naming conventions. No edit to address this will be made as it poses no functional threat to the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	SDG.sol#L766,878,942
Status	Acknowledged

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.

```
totalShares = totalShares.sub(shares[shareholder].amount).add(amount)
launchedAt = launched_
transferBlockTime += _addSeconds
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	SDG.sol#L200,280,310,351
Status	Acknowledged

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 i
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	SDG.sol#L934
Status	Acknowledged

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.

```
IERC20(_token).transfer(owner, _amount)
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

No edit will be made to address this.

Functions Analysis

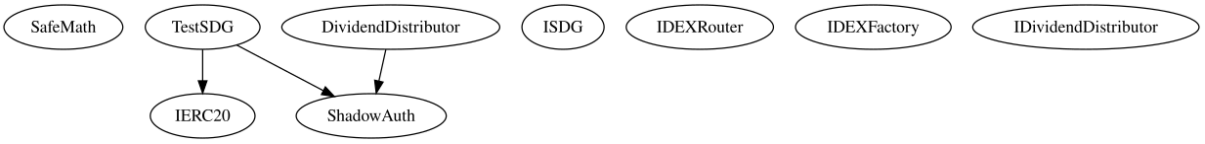
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ShadowAuth	Implementation			
		Public	✓	-
	authorizeFor	Public	✓	authorizedFor
	authorizeForMultiplePermissions	Public	✓	authorizedFor
	unauthorizeFor	Public	✓	authorizedFor
	unauthorizeForMultiplePermissions	Public	✓	authorizedFor
	isOwner	Public		-
	isAuthorizedFor	Public		-
	isAuthorizedFor	Public		-
	transferOwnership	Public	✓	onlyOwner
	getPermissionNameToIndex	Public		-
	getPermissionUnlockTime	Public		-
	isLocked	Public		-
	lockPermission	Public	✓	authorizedFor
	unlockPermission	Public	✓	-
ShadowGold	Implementation	IERC20, ShadowAuth		
		Public	✓	ShadowAuth
		External	Payable	-

	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	checkTxLimit	Internal		
	processFee	Internal	✓	
	takeFee	Internal	✓	
	checkLaunched	External		-
	launched	Internal		
	launch	Internal	✓	
	setShare	External	✓	-
	getShare	External		-
	getTotalShares	External		-
	getTotalHolderCount	External		-

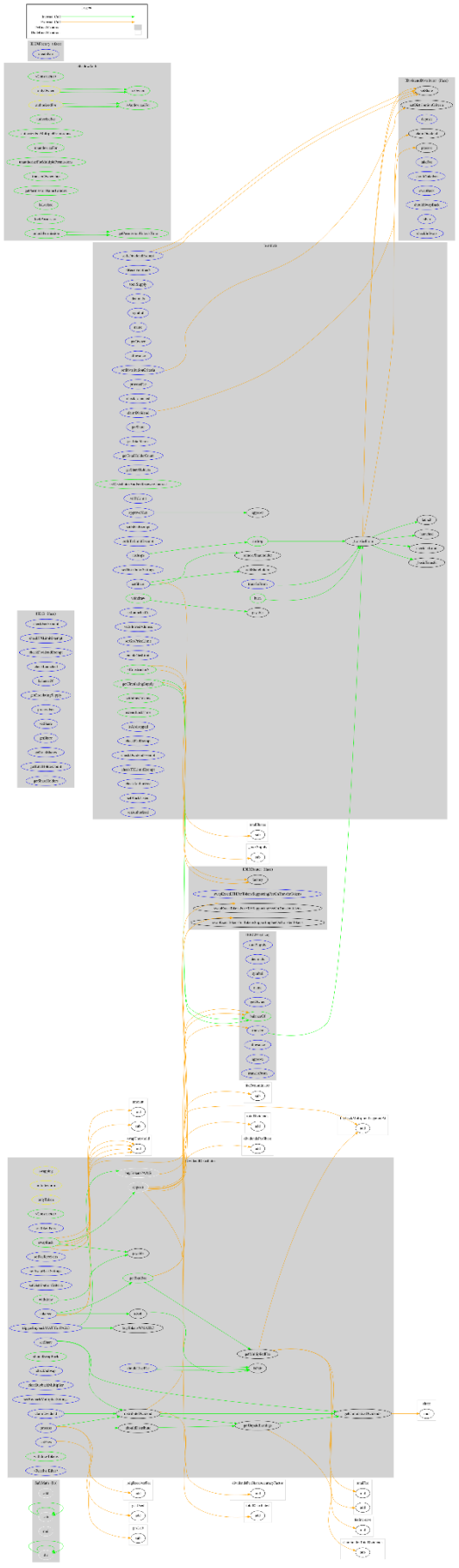
	getShareHolders	External		-
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
	setDistributorAndFeeReceiver	Public	✓	onlyOwner
	setTxLimit	External	✓	authorizedFor
	setIsDividendExempt	External	✓	-
	setIsFeeExempt	External	✓	-
	setIsTxLimitExempt	External	✓	-
	setDistributionCriteria	External	✓	authorizedFor
	setDistributorSettings	External	✓	authorizedFor
	getCirculatingSupply	Public		-
	claimDividend	External	✓	-
	setLaunchedAt	External	✓	authorizedFor
	setAllowedAddress	External	✓	onlyOwner
	setGasPriceLimit	External	✓	onlyOwner
	enableGasLimit	External	✓	onlyOwner
	burn	Public	✓	-
	airdrop	Public	✓	onlyOwner
	airdrops	External	✓	onlyOwner
	withdraw	Public	✓	onlyOwner
	withdrawTokens	Public	✓	onlyOwner
	extendLockTime	Public	✓	onlyOwner
	isAirdropped	External		-

	checkFeeExempt	External		-
	checkDividendExempt	External		-
	checkTXLimitExempt	External		-
	checkAuthorized	External		-
	setBlackListed	External	✓	onlyOwner
	setAuthorized	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

ShadowGold 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 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. The team has acknowledged the findings.

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