Cyberscope

A TAC Security Company

Audit Report

Crypto One

October 2025

Repository	https://github.com/Vsc-blockchain/crypto-one-token
Commit	3a45faa7cf7fc6c7f5df771b0ec7bbb633defc0b
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
●	DPI	Direct Pair Interactions	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	AME	Address Manipulation Exploit	Unresolved
●	IDH	Inconsistent Dividend Handling	Unresolved
●	LRF	Liquidity Rate Frontrun	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	EFC	ERC20 Functionality Constraints	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MAC	Missing Access Control	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MSC	Missing Sanity Check	Unresolved

●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RCS	Redundant Conditional Statements	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/Vsc-blockchain/crypto-one-token
Commit	3a45faa7cf7fc6c7f5df771b0ec7bbb633defc0b
Badge Eligibility	Must Fix Criticals

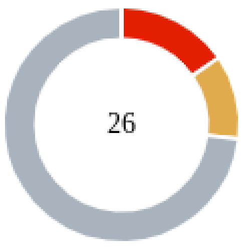
Audit Updates

Initial Audit	30 Sep 2025

Source Files

Filename	SHA256
CryptoOne.sol	9d4ae44612a68497da0f135497c4339daf388f8da4f35aa88c61cb0ca33df997

Findings Breakdown



🟥	Critical	4
🟧	Medium	3
●	Minor / Informative	19

Severity		Unresolved	Acknowledged	Resolved	Other
🟥	Critical	4	0	0	0
🟧	Medium	3	0	0	0
●	Minor / Informative	19	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CryptoOne.sol#L147
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users. The owner may achieve that as outlined in the finding `PTRP`.

Shell
```
function _distributeCollector(uint256 amount)
internal {
if (amount > 0 && _taxCollector != address(0)) {
(bool sent, ) = _taxCollector.call{value:
amount}("");
require(sent, "Native transfer to collector
failed");
}

}
```

Recommendation

The team should follow the recommendations of the `PTRP` finding. Additionally, 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.

DPI - Direct Pair Interactions

Criticality	Critical
Location	CryptoOne.sol#L138
Status	Unresolved

Description

The contract performs DeFi operations such as liquidity provision and token swaps by interacting directly with the liquidity pair, bypassing the standard swap router provided by decentralized exchange (DEX) applications. This approach introduces unnecessary complexity, hinders code readability and introduces errors in the execution flow of decentralized operations such as the removal of liquidity.

Shell
```
else if (_deployed && !swapping && _isPool(to) &&
!_isExempted(from)) {
uint256 taxedValue = value - sellTaxV;
swapping = true;
super._update(from, _pair, sellTaxV);
_correctPoints(from, _pair, sellTaxV);
uint256 out1 =
ISwapBackRouter(_swapBackRouter).swapBack(address(this),
sellTaxV);
uint256 out2 =
ISwapBackRouter(_swapBackRouter).swapBackFromBuys(address(t
his));
swapping = false;

}
```

Recommendation

The team is encouraged to implement the standard router interface to ensure compatibility with decentralized protocols. Specifically, the team is advised to review the documentation of periphery contracts for defi protocols, which outline best practices and integration standards.

UAR - Unexcluded Address Restrictions

Criticality	Critical
Location	CryptoOne.sol#L116,133
Status	Unresolved

Description

The contract enforces operational restrictions on transactions, which may disrupt seamless integration with decentralized applications (dApps) such as exchanges, launchpads, presales, lockers, or staking platforms. These external services often require exemption from certain limitations, such as transaction fees, to function correctly. However, while the contract includes an `exempted` mapping to manage such exemptions, it does not provide a setter function to add addresses to this list. As a result, dApps that depend on being exempt from these restrictions may encounter failures when interacting with the contract.

Shell
```
function _isExempted(address addr) internal view returns
(bool) {return exempted[addr];}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

UTPD - Unverified Third Party Dependencies

Criticality	Critical
Location	CryptoOne.sol#L56,123,124,140,141,245,264,366,367,369,370
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```shell
_swapBackRouter = swapBackRouter;
```

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.

AME - Address Manipulation Exploit

Criticality	Medium
Location	CryptoOne.sol#L322
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This design introduces a significant security risk, as attackers could supply addresses that point to malicious contracts. Specifically, the `transferFrom` function performs low-level calls to the `from` and `to` addresses without validation, enabling potential execution of arbitrary code.

If either address is controlled by an attacker, these external calls could trigger malicious behavior, leading to unauthorized actions or exploitation of the contract. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts.

Shell
```
function transferFrom(address from, address to, uint256
value) public override returns (bool) {
...
if (claim_from > 0) {
(bool sent, ) = from.call{value: claim_from}("");
require(sent, "Native transfer failed");
}

// dividend cashout to "to" before transfer
uint256 claim_to = _prepareCollect(to);
if (claim_to > 0) {
(bool sent, ) = to.call{value: claim_to}("");
require(sent, "Native transfer failed");
}
return true;

}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

IDH - Inconsistent Dividend Handling

Criticality	Medium
Location	CryptoOne.sol#L289,303,322
Status	Unresolved

Description

The contract maintains both a global `unclaimedDividends` value and individual user `_withdrawableDividend` amounts. During the execution of the `transferFrom` function, if a user's `_withdrawableDividend` exceeds the total `unclaimedDividends`, the contract sets `unclaimedDividends` to zero but still returns the full `_withdrawableDividend` amount. This returned value is then used in the transfer logic, which may attempt to transfer more tokens than the contract actually holds. As a result, such transfers are likely to revert due to insufficient token balance.

Shell
```
function _prepareCollect(address account) internal returns
(uint256) {
uint256 _withdrawableDividend =
withdrawableDividendsOf(account);
if (_withdrawableDividend > 0) {
withdrawnDividends[account] = withdrawnDividends[account] +
_withdrawableDividend;
if ( unclaimedDividends >= _withdrawableDividend ) {
unclaimedDividends -= _withdrawableDividend;
} else {
unclaimedDividends = 0;
}
emit DividendsWithdrawn(account, _withdrawableDividend);
}
return _withdrawableDividend;}
```

Recommendation

The team is advised to handle all edge cases gracefully. Specifically, when the contract's balance is insufficient, it should distribute only the available dividends while ensuring the internal state remains consistent.

LRF - Liquidity Rate Frontrun

Criticality	Medium
Location	CryptoOne.sol#L52,181
Status	Unresolved

Description

The contract initializes a new token pair during a call to the initialize method and permits token transfers before liquidity is added. In the `deployDex` function, the admin is responsible for supplying initial liquidity to the pair. However, if the token is already accessible to the public, a third party could front-run the admin by adding liquidity first at a manipulated ratio. This could result in an unfavorable exchange rate, potentially disrupting decentralized applications like presale contracts or creating unsustainable liquidity requirements.

Shell
```
_pair = _createPair(address(this), _weth);
```

Recommendation

The team is advised to account for scenarios where liquidity has not yet been provided. Specifically, they should consider either initializing the trading pair and supplying liquidity during deployment or, alternatively, restricting contract operations until liquidity is provided. The latter should be executed before any third-party integrations with decentralized applications can be assumed.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	CryptoOne.sol#L96,256
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

Shell
```
pointsCorrection[from] += i;

pointsCorrection[to]   -= i;
```

Shell
```
pointsPerShare = pointsPerShare + (amount *
POINTS_MULTIPLIER / shares);
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CryptoOne.sol#L165,172,187,227,376
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Shell
```
function setBuyTax(uint256 newBuyTax) external {...}
function setSellTax(uint256 newSellTax) external {...}
function deployDex() external returns (address) {...}
function addPool(address pool) external {...}
function withdrawTokensFromSwapBackRouterToPairFull()
external returns (uint256) {...}

function setCollector(address _collector) external {...}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	CryptoOne.sol#L80
Status	Unresolved

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.

Shell
```
return (x * y) / 1e18;
```

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.

EFC - ERC20 Functionality Constraints

Criticality	Minor / Informative
Location	CryptoOne.sol#L79,86,92,94,254,268,271,332,339
Status	Unresolved

Description

The contract includes multiple conditional statements within its transaction flow that may disrupt user transactions id executed. The contract must be able to tolerate such scenarios by ensuring that transfer methods can execute reliably under all conditions, including those described by the finding `AAO`.

Shell
```
require(shares > 0, "cannot distribute if total
supply is 0");
```

Recommendation

The contract implements a token mechanism. It is critical that all additional functionalities of the token do not compromise its core operations. The team must ensure that, under all conditions, the token can reliably perform its fundamental functions as defined by the ERC20 standard, including transfers and balance updates, while maintaining its consistency.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	CryptoOne.sol#L55
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Shell
```
_deployed = false;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MAC - Missing Access Control

Criticality	Minor / Informative
Location	CryptoOne.sol#L43
Status	Unresolved

Description

The `initialize` functions can be frontrun during deployment, allowing administrative roles to be transferred to third parties not associated with the team. Such third parties would gain access to all the functions of the system.

```
Shell
function initialize(uint256 initSupply, address taxAdmin,
address factory, address weth, address swapBackRouter)
public initializer {
...

}
```

Recommendation

The team is advised to implement proper access controls to ensure that only authorized team members can call these functions.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	CryptoOne.sol#L226,375
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.

Shell
```
function addPool(address pool) external {...}

function setCollector(address _collector) external
{...}
```

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.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	CryptoOne.sol#L48,49,50
Status	Unresolved

Description

The contract does not adequately verify the initialized variables in the initialize method. If the variables are not initialized correctly, the contract may not function as intended.

```
Shell
_tax_admin = taxAdmin;
_factory = factory;

_weth = weth;
```

Recommendation

It is recommended that the contract implements proper sanity checks to ensure it operates as intended.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	CryptoOne.sol#L140,141
Status	Unresolved

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.

Shell
```
uint256 out1 =
IswapBackRouter(_swapBackRouter).swapBack(address(this),
sellTaxV);

uint256 out2 =
ISwapBackRouter(_swapBackRouter).swapBackFromBuys(address(t
his));
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	CryptoOne.sol#L103
Status	Unresolved

Description

The contract sends funds to a `_taxCollector` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

Shell

```
(bool sent, ) = _taxCollector.call{value:
amount}("");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses and by sending the funds in a non-revertable way.

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	CryptoOne.sol#L246
Status	Unresolved

Description

The contract contains redundant conditional statements that can be simplified to improve code efficiency and performance. Conditional statements that always resolve as true are unnecessary and lead to larger code size, increased memory usage, and slower execution times. By eliminating redundancies, the code can be made more concise and efficient, reducing gas costs and improving runtime performance.

Shell
```
if (address(this) != address(0)) shares -=
balanceOf(address(this));
```

Recommendation

It is recommended to refactor conditional statements by eliminating unnecessary code structures. This practice minimizes the number of operations required, reduces the code footprint, and optimizes memory and gas usage. Simplifying such statements makes the code more readable and improves its overall performance.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	CryptoOne.sol#L140,141
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

Shell
```
uint256 out1 =
ISwapBackRouter(_swapBackRouter).swapBack(address(this),
sellTaxV);

uint256 out2 =
ISwapBackRouter(_swapBackRouter).swapBackFromBuys(address(t
his));
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	CryptoOne.sol#L195,202
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

Shell
```
uint256 wethBalance =
IERC20(_weth).balanceOf(address(this));

uint256 tokenBalance =

this.balanceOf(address(this));
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CryptoOne.sol#L24,26,27,28,29,375
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.

Shell
```
address private _tax_admin
address public _factory
address public _weth
address public _pair
address public _swapBackRouter

address _collector
```

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/stable/style-guide.html#naming-conventions.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CryptoOne.sol#L84
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

Shell
```
function _divDecimal(uint256 x, uint256 y) internal pure
returns (uint256) {
require(y != 0, "Division by zero");
return (x * 1e18) / y;

}
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CryptoOne.sol#L43
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.

Shell
```
address taxAdmin
address factory

address weth
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CryptoOne.sol#L48,49,50,56,331,338,377
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.

Shell
```
_tax_admin = taxAdmin
_factory = factory
_weth = weth
_swapBackRouter = swapBackRouter
(bool sent, ) = from.call{value: claim_from}("")
(bool sent, ) = to.call{value: claim_to}("")

_taxCollector = _collector
```

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	CryptoOne.sol#L2
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.

Shell
```
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	CryptoOne.sol#L208,209
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.

Shell
```
IERC20(_weth).transfer(address(pairContract),
wethBalance)

IERC20(address(this)).transfer(address(pairContrac
t), tokenBalance)
```

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**
IWETH	Interface			
	withdraw	External	✓	-
CryptoOne	Implementation	Initializable, ERC20Upgradeable		
	initialize	Public	✓	initializer
	_createPair	Internal	✓	
	_isPool	Internal		
	_isExempted	Internal		
	_mulDecimal	Internal		
	_divDecimal	Internal		
	_correctPoints	Internal	✓	
	_distributeCollector	Internal	✓	
	_update	Internal	✓	
	setBuyTax	External	✓	-
	setSellTax	External	✓	-
	buyTax	External		-
	sellTax	External		-
	deployDex	External	✓	-
	addPool	External	✓	-

	_totalShares	Internal		
	_distributeDividends	Internal	✓	
	cumulativeDividendsOf	Public		-
	withdrawableDividendsOf	Public		-
	withdrawnDividendsOf	Public		-
	_prepareCollect	Internal	✓	
	withdrawDividends	External	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	taxAdmin	External		-
	collector	External		-
	factory	External		-
	weth	External		-
	deployed	External		-
	withdrawTokensFromSwapBackRouterToPairFull	External	✓	-
	setCollector	External	✓	-
		External	Payable	-

Summary

Crypto One 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. There is also a limit of max 20% fees.

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 TAC 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.

A *TAC Security* Company

The Cyberscope team

cyberscope.io