



Cyberscope

A *TAC Security* Company

Audit Report

Zypto

February 2026

Network BASE

Address 0xdE184C7228430CCo03A4A5792234A6fc99728EF1

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	SemiResolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RED	Redundant Event Declaration	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PISV	Potentially Inconsistent State Variable	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	PZAS	Possible Zero Amount Swap	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Risk Classification	5
Review	6
Audit Updates	6
Source Files	7
Findings Breakdown	8
ST - Stops Transactions	9
Description	9
Recommendation	9
MT - Mints Tokens	10
Description	10
Recommendation	10
BT - Burns Tokens	12
Description	12
Recommendation	12
RED - Redundant Event Declaration	14
Description	14
Recommendation	14
PLPI - Potential Liquidity Provision Inadequacy	15
Description	15
Recommendation	17
CCR - Contract Centralization Risk	18
Description	18
Recommendation	19
PTRP - Potential Transfer Revert Propagation	20
Description	20
Recommendation	21
PISV - Potentially Inconsistent State Variable	22
Description	22
Recommendation	23
PVC - Price Volatility Concern	24
Description	24
Recommendation	24
PZAS - Possible Zero Amount Swap	25
Description	25
Recommendation	26
L04 - Conformance to Solidity Naming Conventions	27

Description	27
Recommendation	28
L16 - Validate Variable Setters	29
Description	29
Recommendation	29
Functions Analysis	30
Inheritance Graph	31
Flow Graph	32
Summary	33
Disclaimer	34
About Cyberscope	35

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

Contract Name	ZyptoToken
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://basescan.org/address/0xde184c7228430cca03a4a5792234a6fc99728ef1
Address	0xde184c7228430cca03a4a5792234a6fc99728ef1
Network	BASE
Symbol	ZYPTO
Decimals	18

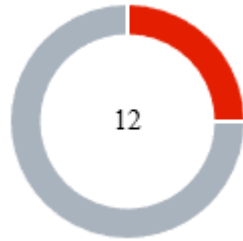
Audit Updates

Initial Audit	20 Feb 2026 https://github.com/cyberscope-io/audits/blob/main/1-zypto/v1/audit.pdf
Corrected Phase 2	23 Feb 2026 https://github.com/cyberscope-io/audits/blob/main/1-zypto/v2/audit.pdf
Corrected Phase 3	25 Feb 2026 https://github.com/cyberscope-io/audits/blob/main/1-zypto/v3/audit.pdf
Corrected Phase 4	26 Feb 2026

Source Files

Filename	SHA256
ZyptoToken.sol	c5beaa73c0b645cd6f966a5446ab59a5978d8780e37af12545cbce2823 374d5e

Findings Breakdown



● Critical	3
● Medium	0
● Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	3	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Zypto.sol#L130,198
Status	SemiResolved

Description

The owner has enabled the transactions for all users by calling the `openTrade` function, which is irreversible. However, sell transactions can still be halted if the internal `_swapAndLiquify` function reverts during execution, as described by the findings PLPI, PTRP & PZAS.

Shell

```
function openTrade() external onlyOwner {
    _isOpenForTrading = true;
}

if (isSell && _swapAndLiquifyEnabled &&
    !_inSwapAndLiquify) {
    uint256 contractBalance =
    balanceOf(address(this));

    if (contractBalance >= _swapTokensAtAmount) {
        _swapAndLiquify(contractBalance);
    }
}
```

Recommendation

The team should ensure that the internal swap logic cannot revert and block sell transactions. This can be achieved by executing the swap functionality in a non-revertable manner and by adding appropriate validation checks to prevent zero-amount operations. Additionally, we strongly recommend a multi-signature wallet to manage the contract's admin functions.

MT - Mints Tokens

Criticality	Critical
Location	Zypto.sol#L118
Status	Unresolved

Description

The contract owner and the `tokenPool` address have the authority to mint tokens. Both entities can achieve that by calling the `mint` function. As a result, the contract tokens will be highly inflated. Furthermore, the owner can set `tokenPool` to an arbitrary address using the `setTokenPool` method.

Shell

```
function mint(address account, uint256 amount) external {
    require(msg.sender == tokenPool || msg.sender ==
owner(), "Not authorized");
    _mint(account, 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.
- Setting the `tokenPoo1` to an immutable value once defined.

BT - Burns Tokens

Criticality	Critical
Location	Zypto.sol#L123
Status	Unresolved

Description

The contract owner and the tokenPool address have the authority to burn tokens from a specific address. Both entities can achieve that by calling the `burn` function. As a result, the targeted address will lose the corresponding tokens. Furthermore, the owner can set tokenPool to an arbitrary address using the `setTokenPool` method.

Shell

```
function burn(address account, uint256 amount) external {
    require(msg.sender == tokenPool || msg.sender ==
owner(), "Not authorized");
    _burn(account, 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.
- Setting the tokenPool to an immutable value once defined.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	ZyptoToken.sol#L75
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

Shell

```
event ExcludeMultipleAccountsFromFees(address[]  
accounts, bool isExcluded);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Zypto.sol# L223,248
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.

Furthermore, the owner has the ability to designate arbitrary addresses as Automated Market Maker pairs. Transfers involving any address flagged as an AMM pair are treated as buy/sell operations and may trigger the internal swap-and-liquify logic. Since these addresses may not be legitimate DEX pairs, triggering a swap during buys from a non-legitimate DEX pair to the actual pair can cause the router interaction to fail and revert.

Shell

```
function _swapTokensForETH(uint256 tokenAmount, address to)
private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = _WETH;

    _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount, 0, path, to, block.timestamp
    );
}

function _swapAndLiquify(uint256 tokenAmount) private
lockTheSwap {
    if (_pendingMarketingValue > 0) {
        uint256 mAmt = _pendingMarketingValue > tokenAmount
? tokenAmount : _pendingMarketingValue;
        _swapTokensForETH(mAmt, _marketingWallet);
        _pendingMarketingValue -= mAmt;
        tokenAmount = balanceOf(address(this));
    }

    uint256 half = tokenAmount / 2;
    uint256 otherHalf = tokenAmount - half;
    uint256 initialETH = address(this).balance;

    _swapTokensForETH(half, address(this));

    uint256 newETH = address(this).balance - initialETH;
    if (newETH > 0 && otherHalf > 0) {
        _uniswapV2Router.addLiquidityETH{value: newETH}(
            address(this), otherHalf, 0, 0, owner(),
            block.timestamp
        );
    }
    emit SwapAndLiquify(half, newETH);
}
```

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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Zypto.sol#L112,118,123,130,135,140,145,153,161,166,172
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 setTokenPool(address _tokenPool) external
onlyOwner {...}
function mint(address account, uint256 amount) external
{...}
function burn(address account, uint256 amount) external
{...}
function openTrade() external onlyOwner {...}
function setSwapAmount(uint256 swapTokensAtAmount) public
onlyOwner {...}
function setAutomatedMarketMakerPair(address pair, bool
value) external onlyOwner {...}
function setBuyFee(uint256 liquidityFee, uint256
marketingFee) public onlyOwner {...}
function setSwapAndLiquifyEnabled(bool enabled) public
onlyOwner {...}
function excludeFromFees(address account, bool excluded)
public onlyOwner {...}
function setMarketingWallet(address account) public
onlyOwner {...}
function setSellFee(uint256 liquidityFee, uint256
marketingFee) public onlyOwner {...}
```

```
function setMarketingWallet(address account) public  
onlyOwner {...}
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Zypto.sol#L166,252
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` 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
_uniswapV2Router.swapExactTokensForETHSupportingFe
eOnTransferTokens(
    tokenAmount, 0, path, to,
    block.timestamp
);
```

```
Shell
function setMarketingWallet(address account)
public onlyOwner {
    _marketingWallet = payable(account);
    _isExcludedFromFees[account] = true;
    emit SetMarketingWallet(account);
}
```

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 or by sending the funds in a non-revertable way, including try-catch schemes.

PISV - Potentially Inconsistent State Variable

Criticality	Minor / Informative
Location	Zypto.sol#L259
Status	Unresolved

Description

The contract provides a `rescueERC20` function that allows the owner to withdraw the entire balance of any ERC20 token held by the contract. However, if the rescued token is the contract's own token (`address(this)`), the `_pendingMarketingValue` state variable is never reset to zero. This creates an accounting inconsistency where the variable continues to track a marketing allocation that no longer exists in the contract's balance.

When fees subsequently accumulate and `_swapAndLiquify` is triggered, the stale `_pendingMarketingValue` will cause a disproportionate amount of tokens to be allocated toward the marketing swap, effectively shortchanging the liquidity provision.

```
Shell
function rescueERC20(IERC20 _token) external
onlyOwner {
    uint256 balance =
_token.balanceOf(address(this));
    _token.safeTransfer(msg.sender, balance);
    emit EmergencyRescueERC20(address(_token),
balance);
}
```

Recommendation

The contract should reset `_pendingMarketingValue` to zero when the rescued token is the contract's own token. Alternatively, the contract could restrict `rescueERC20` from withdrawing its own token entirely, limiting the function to recovering only third-party tokens accidentally sent to the contract.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	ZyptoToken.sol#L135
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_swapTokensAtAmount` 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.

```
Shell
function setSwapAmount(uint256 swapTokensAtAmount)
public onlyOwner {
    _swapTokensAtAmount = swapTokensAtAmount;
    emit
    SetSwapTokensAtAmount(swapTokensAtAmount);
}
```

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.

PZAS - Possible Zero Amount Swap

Criticality	Minor / Informative
Location	ZyptoToken.sol# L227,237
Status	Unresolved

Description

The `_swapAndLiquify` function verifies that `_pendingMarketingValue` is greater than zero before executing a swap; however, it does not validate that the provided `tokenAmount` is also greater than zero. As a result, if `_swapTokensAtAmount` is set to zero, the fees are reduced to zero, or the contract's token balance is otherwise drained, the function may attempt to perform swaps with a zero token amount. Most AMM routers reject zero-amount swaps, causing the transaction to revert and effectively blocking all sell transactions.

A similar issue exists during the liquidity portion of the swap. When `_pendingMarketingValue` exceeds `tokenAmount`, the function swaps the entire contract balance for marketing purposes. This may leave the contract with a zero token balance, after which the liquidity logic can attempt additional swaps using a zero amount, leading again to a revert.

Finally, if the contract balance is empty and `_pendingMarketingValue == 0`, the function proceeds to calculate and swap half of the contract's balance for liquidity. If the balance is zero, the contract attempts to execute a zero-value swap, which will revert for the same reasons described above.

Shell

```
function _swapAndLiquify(uint256 tokenAmount) private
lockTheSwap {
    if (_pendingMarketingValue > 0) {
        uint256 mAmt = _pendingMarketingValue > tokenAmount
? tokenAmount : _pendingMarketingValue;
        _swapTokensForETH(mAmt, _marketingWallet);
        _pendingMarketingValue -= mAmt;
        tokenAmount = balanceOf(address(this));
    }

    uint256 half = tokenAmount / 2;
    uint256 otherHalf = tokenAmount - half;
    uint256 initialETH = address(this).balance;

    _swapTokensForETH(half, address(this));

    uint256 newETH = address(this).balance - initialETH;
    if (newETH > 0 && otherHalf > 0) {
        _uniswapV2Router.addLiquidityETH{value: newETH}(
            address(this), otherHalf, 0, 0, owner(),
            block.timestamp
        );
    }
    emit SwapAndLiquify(half, newETH);
}
```

Recommendation

The team should add explicit validation checks to ensure that swaps are only executed when the token amounts involved are greater than zero. Before calling any swap or liquidity logic, the contract should verify that `tokenAmount`, the marketing swap amount, and any liquidity split amounts are non-zero.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ZyptoToken.sol#L40,41,42,46,47,48,49,50,51,52,53,57,59,60,62,63
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
IUniswapV2Router02 public immutable
_uniswapV2Router
address public immutable _WETH
address public immutable _uniswapV2Pair
uint256 public _buyFeeLiquidity
uint256 public _buyFeeMarketing
uint256 public _buyFeeTotal
uint256 public _sellFeeLiquidity
uint256 public _sellFeeMarketing
uint256 public _sellFeeTotal
uint256 public _pendingMarketingValue
address payable public _marketingWallet
bool public _isOpenForTrading = false
bool public _swapAndLiquifyEnabled = true
uint256 public _swapTokensAtAmount = 100_000 *
1e18;
mapping(address => bool) public
_isExcludedFromFees;
mapping(address => bool) public
_isAutomatedMarketMakerPair;

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increases 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>.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ZyptoToken.sol#L113,167
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
tokenPool = _tokenPool
_marketingWallet = payable(account)
```

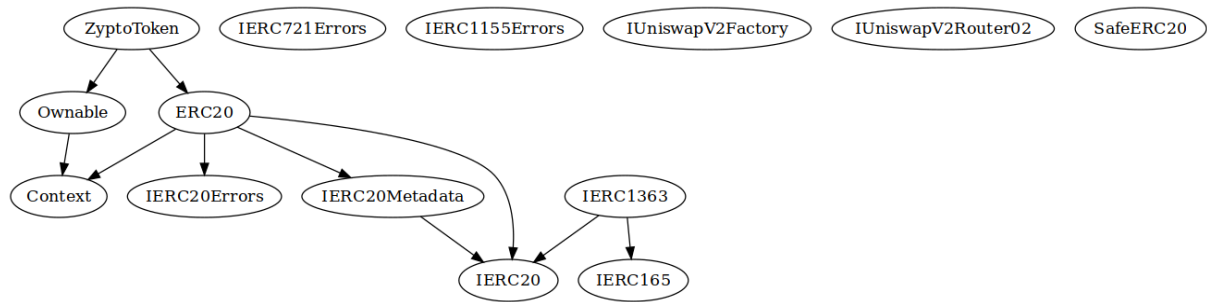
Recommendation

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.

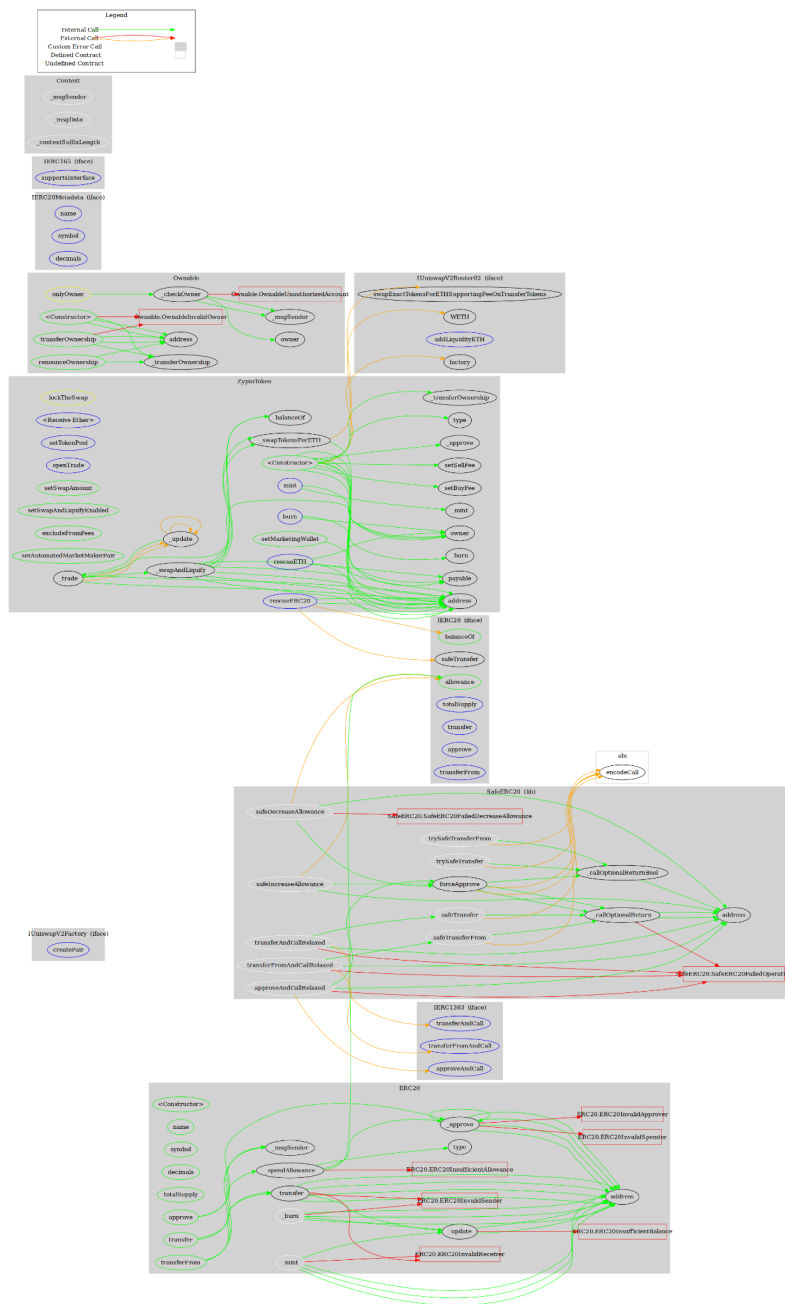
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
ZyptoToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
		External	Payable	-
	setTokenPool	External	✓	onlyOwner
	mint	External	✓	-
	burn	External	✓	-
	openTrade	External	✓	onlyOwner
	setSwapAmount	Public	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setBuyFee	Public	✓	onlyOwner
	setSellFee	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setMarketingWallet	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_update	Internal	✓	
	_trade	Internal	✓	
	_swapAndLiquify	Private	✓	lockTheSwap
	_swapTokensForETH	Private	✓	
	rescueERC20	External	✓	onlyOwner
	rescueETH	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Zypto 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, mint tokens and burn tokens from any address. If the contract owner abuses the mint functionality, then the contract will be highly inflated. If the contract owner abuses the burn functionality, then the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 4% 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