Cyberscope

A TAC Security Company

Audit Report

Quantum Gold

December 2025

Network BSC

Address 0xAF588A95b857c2259F57f810aC9402E52813FE38

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	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	BUIBS	Balance Update Initiated Before Swap	Unresolved
●	PPBE	Potential Pair Balance Extraction	Unresolved
●	PPCM	Potential Profit Calculation Manipulation	Unresolved
●	TAPAI	Transfer And Profit Amount Inconsistencies	Unresolved
●	ILRH	Incorrect Liquidity Removal Handling	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	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

Contract Name	ATV
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://bscscan.com/address/0xaf588a95b857c2259f57f810ac9402e52813fe38
Address	0xaf588a95b857c2259f57f810ac9402e52813fe38
Network	BSC
Symbol	ATV
Decimals	18
Total Supply	11,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	18 Dec 2025

Source Files

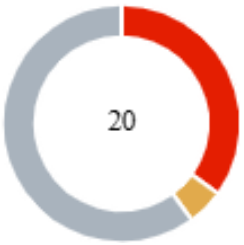
Filename	SHA256
project/contracts/ATV.sol	4abbe1a02ebb8aea281dfc435018ef8b4eec35fed5a633f49372a37379d88591

Overview

Readability Comment

The audit scope is to check for security vulnerabilities, validate the business logic, and propose potential optimizations. The contract does not adhere to best practices for interacting with other smart contracts and decentralized applications (dApps). Specifically, it fails to implement consistent cross-contract interaction patterns, leading to inconsistencies, failed transactions, and overall discrepancies. The development team is strongly advised to follow standardized methodologies for contract interoperability and decentralized application design, ensuring proper transaction flow, error handling, and state consistency. Adopting best practices will enhance the contract's reliability and prevent issues related to failed transactions and broken composability.

Findings Breakdown



Critical	7
Medium	1
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	7	0	0	0
Medium	1	0	0	0
Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	project/contracts/ATV.sol#L722,755,757
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `coldTime` to a large value. As a result, the contract may operate as a honeypot.

```
require(block.timestamp >= lastBuyTime[sender] + coldTime, "cold");
```

Additionally, the trades are initially disabled. The owner can enable trades by setting the `presale` boolean to true.

```
require(presale, “pre”);
```

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.

MT - Mints Tokens

Criticality	Critical
Location	project/contracts/ATV.sol#L1107
Status	Unresolved

Description

The contract `treasury` has the authority to mint tokens. The `treasury` may take advantage of it by calling the `treasuryMint` function. As a result, the contract tokens will be highly inflated.

```
function treasuryMint(address to, uint256 amount) external {
    require(msg.sender == treasury, "!treasury");
    require(to != address(0), "to=0");
    require(amount > 0, "amount=0");
    require(
        treasuryMinted + amount <= TREASURY_MINT_CAP,
        "exceed mint cap"
    );

    treasuryMinted += amount;
    totalSupply += amount;
    unchecked {
        balanceOf[to] += amount;
    }

    emit Transfer(address(0), to, amount);
    emit TreasuryMint(to, amount, treasuryMinted);
}
```

Recommendation

The team should carefully manage the private keys of the `treasury` '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.

BC - Blacklists Addresses

Criticality	Critical
Location	project/contracts/ATV.sol#L1134
Status	Unresolved

Description

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

```
function multi_bclist(
    address[] calldata addresses,
    bool value
) public onlyOwner {
    require(addresses.length < 201);
    for (uint256 i; i < addresses.length; ++i) {
        _rewardList[addresses[i]] = value;
    }
}
```

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.

BUIBS - Balance Update Initiated Before Swap

Criticality	Critical
Location	project/contracts/ATV.sol#L806
Status	Unresolved

Description

The contract is designed to perform a token swap after updating the respective balances. This sequence of operations can cause inconsistencies in the balances used for operations involving the router, which may result in the swap being halted. These inconsistencies disrupt the functionality of the swap, causing transactions to fail and affecting the overall liquidity operations of the contract.

```
if (fee > 0) {
    super._transfer(sender, address(this), fee);
    if (shouldSwapProfit(fee)) {
        swapProfit(fee);
    }
}
if (
    shouldSwapTokenForFund(
        AmountLPFee + AmountMarketingFee + AmountNodeFee
    )
) {
    swapTokenForFund();
}
```

Recommendation

It is recommended to ensure that the swap is triggered only before the balances are updated. This will prevent inconsistencies in the balances used during router operations, ensuring the swap executes successfully and maintaining smooth token sale functionality.

PPBE - Potential Pair Balance Extraction

Criticality	Critical
Location	project/contracts/ATV.sol#L998,1042
Status	Unresolved

Description

The contract allows a `STAKING` address set by the owner to transfer up to 1/3 of the pair contract tokens and send it to the `STAKING` . As a result, this function can break the pool's invariant, distort the market price, and/or remove meaningful liquidity value from LPs/traders.

```
function recycle(uint256 amount) external {
    require(STAKING == msg.sender, "cycle");
    uint256 maxBurn = balanceOf[uniswapV2Pair] / 3;
    uint256 burn_maount = amount >= maxBurn ? maxBurn : amount;
    super._transfer(uniswapV2Pair, STAKING, burn_maount);
    IUniswapV2Pair(uniswapV2Pair).sync();
}
```

The case is also similar for the `dailyBurn` function.

```
function dailyBurn() external {
    ...
    if (burnAmount > 0) {
        lastDailyBurnTime = uint40(block.timestamp);
        super._transfer(uniswapV2Pair, address(0xdead), burnAmount);
        IUniswapV2Pair(uniswapV2Pair).sync();
    }
    ...
}
```

Recommendation

It is recommended to remove or fundamentally redesign this mechanism so that tokens are never transferred directly from the liquidity pair address through privileged contract logic, as this violates AMM assumptions and exposes liquidity providers to severe value loss.

PPCM - Potential Profit Calculation Manipulation

Criticality	Critical
Location	project/contracts/ATV.sol#L809
Status	Unresolved

Description

The contract determines whether a seller is “in profit” and how much additional profit-based fee should be applied by relying on spot price quotations derived from the liquidity pool reserves at the moment of the sell. These quotations are obtained via routing logic that reflects the current state of the pool, not a time-averaged or execution-weighted price. As a result, the profit calculation is directly coupled to transient reserve conditions that can change within the same block. This creates a structural weakness where economically meaningful decisions depend on values that are not stable or resistant to manipulation.

An attacker front-runs a user’s sell by trading against the ATV/USDT pool to move the price in a direction that distorts the profit calculation. The victim’s sell then executes under these manipulated conditions, resulting in inflated or reduced profit-based fees. Finally, the attacker back-runs the transaction to restore the pool price and capture arbitrage gains. Compared to a standard sandwich attack, the impact is amplified because the manipulated price not only worsens execution but also directly affects the contract’s internal profit-fee logic.

```
uint256 fee;
if (tOwnedU[sender] >= amountUOut) {
    // 仍在成本内：不收盈利税，仅减少成本基数
    unchecked {
        tOwnedU[sender] = tOwnedU[sender] - amountUOut;
    }
} else if (tOwnedU[sender] > 0 && tOwnedU[sender] < amountUOut) {
    // 部分超过成本：对"利润"部分折算成ATV后收15%
    uint256 profitU = amountUOut - tOwnedU[sender];
    uint256 profitThis = Helper.getAmountOut(
        profitU,
        reserveU,
        reserveThis
    );
    fee = (profitThis * 15) / 100; // 15%
    tOwnedU[sender] = 0; // 成本用尽
} else {
    // 没有成本记录：按本次卖出量的15%作为盈利税
    fee = (amount * 15) / 100;
    tOwnedU[sender] = 0;
}

// 收取盈利税并视情况立即换成 USDT 分账
if (fee > 0) {
    super._transfer(sender, address(this), fee);
    if (shouldSwapProfit(fee)) {
        swapProfit(fee);
    }
}
```

Recommendation

Profit-based fee calculations should not rely on instantaneous spot prices derived from liquidity pool reserves. To mitigate manipulation risks, the design should incorporate a profit accounting model that does not depend on mutable pool state at the moment of transfer.

TAPAI - Transfer And Profit Amount Inconsistencies

Criticality	Critical
Location	project/contracts/ATV.sol#L827
Status	Unresolved

Description

During a sell, the contract calculates the amount of profit a user will receive. The contract then subtracts fees from the profit. However, by subtracting these fees, the contract dynamically changes the profit the user will receive and since the swap amount are not calculated proportionally, this will result in incorrect state updates as well as lower than expected profits.

```
uint256 fee;
if (tOwnedU[sender] >= amountUOut) {
    // 仍在成本内：不收盈利税，仅减少成本基数
    unchecked {
        tOwnedU[sender] = tOwnedU[sender] - amountUOut;
    }
} else if (tOwnedU[sender] > 0 && tOwnedU[sender] < amountUOut) {
    // 部分超过成本：对"利润"部分折算成ATV后收15%
    uint256 profitU = amountUOut - tOwnedU[sender];
    uint256 profitThis = Helper.getAmountOut(
        profitU,
        reserveU,
        reserveThis
    );
    fee = (profitThis * 15) / 100; // 15%
    tOwnedU[sender] = 0; // 成本用尽
} else {
    // 没有成本记录：按本次卖出量的15%作为盈利税
    fee = (amount * 15) / 100;
    tOwnedU[sender] = 0;
}

// 收取盈利税并视情况立即换成 USDT 分账
if (fee > 0) {
    super._transfer(sender, address(this), fee);
    if (shouldSwapProfit(fee)) {
        swapProfit(fee);
    }
}

...

super._transfer(sender, recipient, amount - fee - marketingFee);
```

Recommendation

The team should restructure the `update` function to ensure that such inconsistencies are avoided. Additionally, it is advisable that fees are extracted after the swaps are performed.

ILRH - Incorrect Liquidity Removal Handling

Criticality	Medium
Location	project/contracts/ATV.sol#L720
Status	Unresolved

Description

In the update function, if the `uniswapV2Pair` is the `sender` address, the transfer is automatically considered a buy. However, during the removal of liquidity, the `sender` address will also be the pair. In that case, the contract will treat this transfer as a buy, and it will update the pool's local reserve.

```
if (uniswapV2Pair == sender) {
    /// ============ 买入路径（从交易对买到 ATV） ============
    require(presale, "pre");
    unchecked {
        // 读池子储备并限制单笔买入不超过代币储备的 10%
        (uint112 reserveU, uint112 reserveThis, ) = IUniswapV2Pair(
            uniswapV2Pair
        ).getReserves();
        require(amount <= reserveThis / 10, "max cap buy"); // 10%
        updatePoolReserve(reserveU);

        // 记录买家“投入的 USDT 成本基数”（用于日后卖出盈利税比较）
        uint256 amountUBuy = Helper.getAmountIn(
            amount,
            reserveU,
            reserveThis
        );
        tOwnedU[recipient] = tOwnedU[recipient] + amountUBuy;
        lastBuyTime[recipient] = uint40(block.timestamp);

        // 节点分红费: 1.5%
        uint256 nodeFee = (amount * 15) / 1000;
        AmountNodeFee += nodeFee;
        super._transfer(sender, address(this), nodeFee);

        // LP费: 1.5%
        uint256 LPFee = (amount * 15) / 1000;
        AmountLPFee += LPFee;
        super._transfer(sender, address(this), LPFee);

        // 实际到手（扣除3%）
        super._transfer(sender, recipient, amount - nodeFee - LPFee);
    }
}
```

Recommendation

It is advised that the contract does not handle the liquidity removal process as a buy.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L998,1067,1097
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.

```
function recycle(uint256 amount) external {}
function setStaking(address addr) external onlyOwner {}
function setTreasury(address _treasury) external 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.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L1138
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to 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(addresses.length < 201)
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L1056,1062,1068,1074,1080
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.

```
marketingAddress = addr;
profitAddress = addr;
STAKING = addr;
nodeAddress = addr;
levelBonusAddress = addr;
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L984
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.

```
uniswapV2Router
    .swapExactTokensForTokensSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        to,
        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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L1050
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapAtAmount` 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 setSwapAtAmount(uint256 newValue) public onlyOwner {
    swapAtAmount = newValue;
}
```

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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L741,746
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
uint256 nodeFee = (amount * 15) / 1000;
uint256 LPFee = (amount * 15) / 1000;
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L67,69,100,171,572,599,600,608,612,619,654,1097,1134
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
IUniswapV2Router02 constant uniswapV2Router = IUniswapV2Router02(_ROUTER)
uint256 public AmountMarketingFee;
uint256 public AmountLPFee;
mapping(address => bool) public _rewardList;
address public STAKING;
uint256 public AmountNodeFee;
...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L655,1051
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
coldTime = _coldTime;
swapAtAmount = newValue;
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L783
Status	Unresolved

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 fee;
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L417,1056,1062,1068,1074,1080
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.

```
owner = newOwner
marketingAddress = addr
profitAddress = addr
STAKING = addr
nodeAddress = addr
levelBonusAddress = addr
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L371
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	project/contracts/ATV.sol#L908,916,926,947
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.

```
IERC20(USDT).transferFrom(
    address(distributor),
    marketingAddress,
    oneThird
);
IERC20(USDT).transferFrom(
    address(distributor),
    nodeAddress,
    oneThird
);
...
```

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.

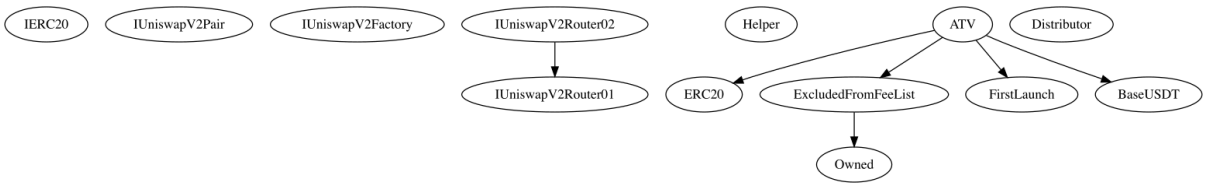
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Owned	Implementation			
		Public	✓	-
	transferOwnership	Public	✓	onlyOwner
ERC20	Implementation			
		Public	✓	-
	approve	Public	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
ExcludedFromFeeList	Implementation	Owned		
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	excludeMultipleAccountsFromFee	Public	✓	onlyOwner
FirstLaunch	Implementation			
	launch	Internal	✓	

Distributor	Implementation			
		Public	✓	-
BaseUSDT	Implementation			
		Public	✓	-
ATV	Implementation	ExcludedFromFeeList, BaseUSDT, FirstLaunch, ERC20		
	setPresale	External	✓	onlyOwner
	setColdTime	External	✓	onlyOwner
	updatePoolReserve	Public	✓	-
	updatePoolReserve	Private	✓	
	getReserveU	External		-
		Public	✓	Owned ERC20
	_transfer	Internal	✓	
	marketingFeeRate	Internal		
	shouldSwapTokenForFund	Internal		
	swapTokenForFund	Internal	✓	lockTheSwap
	shouldSwapProfit	Internal		
	swapProfit	Internal	✓	lockTheSwap
	swapAndLiquify	Internal	✓	
	addLiquidity	Internal	✓	
	swapTokenForUsdt	Internal	✓	

	recycle	External	✓	-
	dailyBurn	External	✓	-
	setSwapAtAmount	Public	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setProfitAddress	External	✓	onlyOwner
	setStaking	External	✓	onlyOwner
	setNodeAddress	External	✓	onlyOwner
	setLevelBonusAddress	External	✓	onlyOwner
	setTreasury	External	✓	onlyOwner
	treasuryMint	External	✓	-
	treasuryMintRemaining	External		-
	multi_bclist	Public	✓	onlyOwner
	isReward	Public		-

Inheritance Graph



Flow Graph

Summary

Quantum Gold 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 stopping transactions, minting tokens, and massively blacklist addresses. If the contract owner abuses the mint functionality, then the contract will be highly inflated. 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 18% sell fees, and the buy fees are fixed at 3%.

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