Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: 6ENSE (6OS)
Website: 6ense.it
Platform: Ethereum
Language: Solidity
Date: March 24th, 2025

Table of contents

Introduction 4

Project Background 4

Key Smart Contract and Exchange Addresses..... 6

Whitelist Address(excluded from fee) 6

Audit Scope 6

Claimed Smart Contract Features 7

Audit Summary 9

Technical Quick Stats 10

Business Risk Analysis 11

Code Quality 12

Documentation 12

Use of Dependencies 12

AS-IS overview 13

Severity Definitions 15

Audit Findings 16

Conclusion 18

Our Methodology 19

Disclaimers 21

Appendix

- Code Flow Diagram 22
- Slither Results Log 23
- Solidity static analysis 26
- Solhint Linter 28

THIS IS A SECURITY AUDIT REPORT DOCUMENT AND MAY CONTAIN INFORMATION THAT IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES THAT CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the 6ENSE team to perform the Security audit of the 6OS Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 24th, 2025.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

The Token6OS contract is an ERC-20 token with reflection mechanics, tax mechanisms, and liquidity management features. It includes:

Key Features:

1. Reflection Mechanism:

- It uses a reflection model, where holders passively earn more tokens.
- Instead of sending rewards manually, the total token supply is split into:
    - _rOwned: Reflects the actual balance for each holder.
    - _tOwned: Stores token amounts before reflections.
- Rewards are distributed automatically whenever transfers occur.

2. Tax System:

- Two main taxes:
    - Reflection Tax: Distributes fees among holders.
    - PlantoGroup Tax: Sends fees to a designated wallet.
- Taxes are adjustable by the owner but capped at 6%.

3. Liquidity Management:

- Integrates Uniswap V2 for automated market-making.
- Contract swaps collect taxes for USDC and send them to the PlantoGroup Wallet.

- Uses a tax threshold to determine when to swap.

4. Transaction Limits & Controls:

- Max Transfer Limit: Prevents large transfers.
- Trading Enable/Disable: The owner can toggle trading.
- Exclusions: Certain addresses can be exempt from fees and reflections.

5. Ownership & Governance:

- Only the owner can:
    - Update taxes, tax threshold, and max transfer amount.
    - Enable or disable trading.
    - Exclude/include addresses from rewards and fees.
    - Update the PlantoGroup Wallet address.

Token6OS is a **reflection-based ERC20 token** with a **built-in tax system, Uniswap liquidity integration, and automated USDC conversion**. It rewards holders passively while collecting fees for external use.

Key Smart Contract and Exchange Addresses:

Owner Address	0x06fE8f07513f7a0C3a3D4A948bee12263c3197AF
Planto Group Wallet	0x9C27bF771ea62dE8e4281e971f72d52D49c58486
Uniswap Pair Address	0xB8Edb72A75ef76bC60FF4D5da124cf118a1Ed8B0
Uniswap Router Address	0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
USDC Address	0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48

Whitelist Address(excluded from fee):

Token Contract Address	0x69119d3BC1277efb4200c12082F46E596FDF9d39
Owner Address	0x06fE8f07513f7a0C3a3D4A948bee12263c3197AF

Audit scope

Name	**Code Review and Security Analysis Report for 6ENSE (6OS) Token Smart Contract**
Platform	**Ethereum**
File	Token6OS.sol
Smart Contract Code	0x69119d3bc1277efb4200c12082f46e596fdf9d39
Audit Date	March 24th, 2025

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: 6ENSE • Symbol: 6OS • Decimals: 18 • Total Supply: 9,630 million $6OS	**YES, This is valid.**
Core Features: **1. Tokenomics & Supply Management:** • Reflection Mechanism: Implements a reflection-based fee distribution system. • Planto Group Fee: A portion of transactions is allocated to a designated wallet. **2. Trading & Transfer Controls :** • Max Transfer Limit: Transactions cannot exceed 192.6 million tokens (modifiable). • Trading Enable/Disable: The owner can toggle trading. • Tax Threshold: Swaps and tax distribution occur when the contract balance exceeds 1,000 tokens. **3. Fee System:** • Reflection Tax: Distributes a percentage of transactions to existing holders. • Planto Group Tax: Sends fees to a designated wallet. • Fee Exclusions: Certain addresses can be excluded from paying fees. **4. Liquidity & Swapping:** • Uniswap Integration: Uses UniswapV2 Router for token swaps.	**YES, This is valid.**

• Swap & Liquify: Automatically convert fees to USDC and send them to a wallet. **5. Reflection & Rewards:** • Reflections: Rewards holders by redistributing transaction fees. • Exclusion from Rewards: Certain addresses can be excluded from receiving reflections. **6. Administrative Controls:** • Ownership Functions: Includes onlyOwner modifiers for sensitive functions. • Adjustable Taxes: Reflection and Planto Group taxes are adjustable (max 6%). • Max Transfer Updates: The Owner can modify max transaction limits.	

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contract is **"Secured."** This token contract has ownership control and is not fully 100% decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 1 very low-level issue.

Investor Advice: The technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	Adjustable (reflection and plantoGroup Tax - max 6%)
Sell Tax	Adjustable (reflection and plantoGroup Tax - max 6%)
Can Buy	Yes
Can Sell	Yes
Max Tax	6%
Modify Tax	Yes
Fee Check	Yes
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	Not Detected
Max Tax?	Yes
Is it Anti-whale?	Yes
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	Not Detected
Can Mint?	No
Is it a Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in 6OS Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the 6OS Token.

The 6ENSE Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a 6OS Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that is based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Token6OS.sol : Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	name	external	Passed	No Issue
3	symbol	external	Passed	No Issue
4	decimals	external	Passed	No Issue
5	totalSupply	external	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	transfer	external	Passed	No Issue
8	allowance	external	Passed	No Issue
9	approve	write	Passed	No Issue
10	transferFrom	external	Passed	No Issue
11	increaseAllowance	external	Passed	No Issue
12	decreaseAllowance	external	Passed	No Issue
13	isExcludedFromReward	external	Passed	No Issue
14	totalFees	external	Passed	No Issue
15	deliver	external	Passed	No Issue
16	reflectionFromToken	external	Passed	No Issue
17	tokenFromReflection	read	Passed	No Issue
18	excludeFromReward	write	access only owner	No Issue
19	includeInReward	external	access only owner	No Issue
20	excludeFromFee	external	access only owner	No Issue
21	includeInFee	external	access only owner	No Issue
22	setplantoGroupWallet	external	access only owner	No Issue
23	updateThreshold	external	access only owner	No Issue
24	receive	external	Passed	No Issue
25	_approve	write	Passed	No Issue
26	_reflectFee	write	Passed	No Issue
27	_takePlantoFee	write	Passed	No Issue
28	_getValues	read	Passed	No Issue
29	_getValue	read	Passed	No Issue
30	_getTValues	read	Passed	No Issue
31	_getRValues	read	Passed	No Issue
32	_getRate	read	Passed	No Issue
33	_getCurrentSupply	read	Passed	No Issue
34	calculateTaxFee	read	Passed	No Issue
35	calculatePlantoTax	read	Passed	No Issue
36	removeAllFee	write	Passed	No Issue
37	setTrading	external	Unused function	Refer Audit Findings
38	updateReflectionTaxPer	external	access only owner	No Issue
39	updatePlantoGroupTax	external	access only owner	No Issue
40	setMaxTransferLimit	external	access only owner	No Issue

41	_transfer	write	Passed	No Issue
42	swapAndLiquify	write	Passed	No Issue
43	swapTokensForUsdc	write	Passed	No Issue
44	_tokenTransfer	write	Passed	No Issue
45	_transferBothExcluded	write	Passed	No Issue
46	_transferStandard	write	Passed	No Issue
47	_transferToExcluded	write	Passed	No Issue
48	_transferFromExcluded	write	Passed	No Issue
49	owner	read	Passed	No Issue
50	onlyOwner	modifier	Passed	No Issue
51	renounceOwnership	write	access only owner	No Issue
52	transferOwnership	write	access only owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Unused function:

The tradeEnabled is not used during transfers.

Resolution: We suggest removing all related codes from the contract. Remove setTrading function, TradeEnabled event, and tradeEnabled boolean variable.

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble.
The following are Admin functions:

Token6OS.sol

- excludeFromReward: Grants the owner the ability to exclude an address from earning reflections.
- includeInReward: Grants the owner the ability to include an account in the reward distribution.
- excludeFromFee: Grants the owner the ability to exclude an address from transaction fees.
- includeInFee: Grants the owner the ability to include an address in transaction fees.
- setplantoGroupWallet: Sets the address of the fund wallet by the owner.
- updateThreshold: Grants the owner the ability to update the threshold amount used for Plantogroup wallet addition.
- setTrading: Enables or disables trading functionality based on the input parameter by the owner.
- updateReflectionTaxPer: Updates the reflection tax (max 6%) by the owner.
- updatePlantoGroupTax: Updates the PlantoGroup tax (max 6%) by the owner.
- setMaxTransferLimit: Sets the maximum buy limit per transaction. Can only be called by the contract owner.

Conclusion

We were given a contract code in the form of Etherscan web links. We have used all possible tests based on the given objects as files. We observed 1 informational severity issue in the smart contract. but this issue is not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract by the best industry practices at the date of this report, about: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

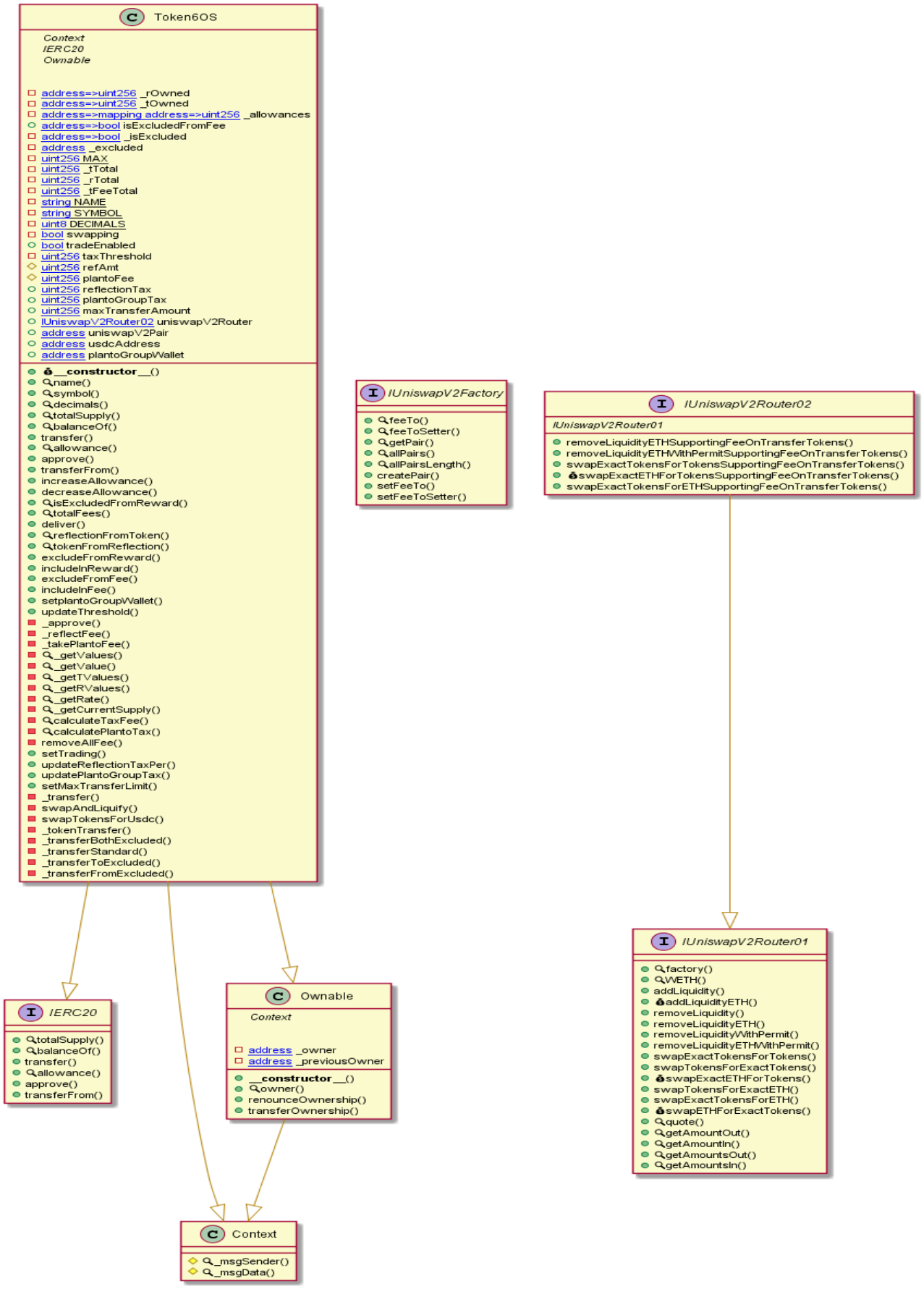
Because the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - 6ENSE (6OS) Token



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We analyzed the project altogether. Below are the results.

Slither Log >> Token6OS.sol

```
INFO:Detectors:
Contract locking ether found:
	Contract Token6OS (Token6OS.sol#338-1116) has payable functions:
	 - Token6OS.receive() (Token6OS.sol#719)
	But does not have a function to withdraw the ether
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Token6OS.allowance(address,address).owner (Token6OS.sol#491) shadows:
	- Ownable.owner() (Token6OS.sol#139-141) (function)
Token6OS._approve(address,address,uint256).owner (Token6OS.sol#733) shadows:
	- Ownable.owner() (Token6OS.sol#139-141) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in Token6OS._transfer(address,address,uint256) (Token6OS.sol#942-980):
	External calls:
	- swapAndLiquify() (Token6OS.sol#962)
		-
uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,plantoGroupWallet,block.ti
mestamp) (Token6OS.sol#1010-1016)
	State variables written after the call(s):
	- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
		- _tFeeTotal = _tFeeTotal + tFee (Token6OS.sol#748)
	- plantoFee = plantoGroupTax (Token6OS.sol#977)
	- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
		- plantoFee = 0 (Token6OS.sol#885)
	- refAmt = reflectionTax (Token6OS.sol#976)
	- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
		- refAmt = 0 (Token6OS.sol#884)
```

```
Reentrancy in Token6OS.transferFrom(address,address,uint256) (Token6OS.sol#518-522):
	External calls:
	- _transfer(sender,recipient,amount) (Token6OS.sol#519)
		-
uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,plantoGroupWallet,block.ti
mestamp) (Token6OS.sol#1010-1016)
	State variables written after the call(s):
	- _approve(sender,_msgSender(),_allowances[sender][_msgSender()] - amount)
(Token6OS.sol#520)
		- _allowances[owner][spender] = amount (Token6OS.sol#737)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Token6OS._transfer(address,address,uint256) (Token6OS.sol#942-980):
	External calls:
	- swapAndLiquify() (Token6OS.sol#962)
		-
uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,plantoGroupWallet,block.ti
mestamp) (Token6OS.sol#1010-1016)
	Event emitted after the call(s):
	- ReflectedFee(tFee) (Token6OS.sol#750)
		- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
	- Transfer(sender,recipient,tTransferAmount) (Token6OS.sol#1077)
		- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
	- Transfer(sender,recipient,tTransferAmount) (Token6OS.sol#1113)
		- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
	- Transfer(sender,recipient,tTransferAmount) (Token6OS.sol#1095)
		- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
	- Transfer(sender,recipient,tTransferAmount) (Token6OS.sol#1060)
		- _tokenTransfer(from,to,amount,takeFee) (Token6OS.sol#979)
Reentrancy in Token6OS.transferFrom(address,address,uint256) (Token6OS.sol#518-522):
	External calls:
	- _transfer(sender,recipient,amount) (Token6OS.sol#519)
		-
uniswapV2Router.swapExactTokensForTokens(tokenAmount,0,path,plantoGroupWallet,block.ti
mestamp) (Token6OS.sol#1010-1016)
	Event emitted after the call(s):
	- Approval(owner,spender,amount) (Token6OS.sol#738)
		- _approve(sender,_msgSender(),_allowances[sender][_msgSender()] - amount)
(Token6OS.sol#520)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Token6OS.includeInReward(address) (Token6OS.sol#648-660) has costly operations inside a
loop:
	- _excluded.pop() (Token6OS.sol#655)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
INFO:Detectors:
Context._msgData() (Token6OS.sol#102-105) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Token6OS._rTotal (Token6OS.sol#349) is set pre-construction with a non-constant function or
state variable:
        - (MAX - (MAX % _tTotal))
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state
INFO:Detectors:
Function IUniswapV2Router01.WETH() (Token6OS.sol#203) is not in mixedCase
Parameter Token6OS.setTrading(bool)._enable (Token6OS.sol#893) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Redundant expression "this (Token6OS.sol#103)" inContext (Token6OS.sol#97-106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Token6OS.setMaxTransferLimit(uint256) (Token6OS.sol#929-933) uses literals with too many
digits:
        - require(bool,string)(amount >= 500000 * 10 ** 18,Max Transfer limit can not be less than
500,000 tokens) (Token6OS.sol#930)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Loop condition i < _excluded.length (Token6OS.sol#846) should use cached array length instead
of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Detectors:
Token6OS._tTotal (Token6OS.sol#348) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-constant
INFO:Slither:Token6OS.sol analyzed (7 contracts with 93 detectors), 18 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

Token6OS.sol

```
Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in Token6OS.(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
Pos: 419:12:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 1035:17:

Gas costs:
Gas requirement of function Token6OS.excludeFromReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 643:93:

Gas costs:
Gas requirement of function Token6OS.includeInReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 668:38:

For loop over a dynamic array:
Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 670:40:

For loop over a dynamic array:
```

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
Pos: 869:12:

ERC20:
ERC20 contract's "decimals" function should have "uint8" as return type
Pos: 465:33:

Similar variable names:
Token6OS.deliver(uint256) : Variables have very similar names "rAmount" and "tAmount". Note: Modifiers are currently not considered by this static analysis.
Pos: 597:27:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 978:5:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 901:0:

Solhint Linter

Solhint Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

Token6OS.sol

```
Compiler version 0.8.26 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:19
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:129
Error message for require is too long
Pos: 9:167
Function name must be in mixedCase
Pos: 5:202
Contract has 18 states declarations but allowed no more than 15
Pos: 1:337
Explicitly mark visibility of state
Pos: 5:360
Explicitly mark visibility of state
Pos: 5:361
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:394
Error message for require is too long
Pos: 9:712
Code contains empty blocks
Pos: 32:718
Error message for require is too long
Pos: 9:942
Error message for require is too long
Pos: 9:943
Error message for require is too long
Pos: 9:944
Avoid making time-based decisions in your business logic
Pos: 13:1014
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io