ASSURE DEFI®
THE VERIFICATION GOLD STANDARD

Security Assessment:
AimBot Token

February 28, 2024

- Audit Status: **Fail**
- Audit Edition: **Standard**

VERIFIED BY ASSURE DEFI ®
INTEGRITY • TRUST • CREDIBILITY

Risk Analysis

Classifications of Manual Risk Results

Classification	Description
🔴 Critical	Danger or Potential Problems.
🟠 High	Be Careful or Fail test.
🟢 Low	Pass, Not-Detected or Safe Item.
ℹ️ Informational	Function Detected

Manual Code Review Risk Results

Contract Privilege	Description
🟢 Buy Tax	5%
🟢 Sale Tax	5%
🟢 Cannot Sale	Pass
🟢 Cannot Sale	Pass
🟢 Max Tax	5%
ℹ️ Modify Tax	Yes
🟢 Fee Check	Pass
🟢 Is Honeypot?	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	Pass
🟢 Pause Transfer?	Not Detected
🟢 Max Tx?	Pass
🟡 Is Anti Whale?	Detected
🟡 Is Anti Bot?	Detected

Contract Privilege	Description
🟢 Is Blacklist?	Not Detected
🟢 Blacklist Check	Pass
🟢 is Whitelist?	Not Detected
🟢 Can Mint?	Pass
🟢 Is Proxy?	Not Detected
🟢 Can Take Ownership?	Not Detected
🟢 Hidden Owner?	Not Detected
ℹ Owner	0x077905FA422A6C1f45Ad81D305e15dD94f8af56E
🟢 Self Destruct?	Not Detected
🟢 External Call?	Detected
🟢 Other?	Not Detected
🟢 Holders	3,702
🟡 Auditor Confidence	Medium Risk
🟡 KYC Present	No
🟡 KYC URL	

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.

Project Overview

Token Summary

Parameter	Result
Address	0x0c48250Eb1f29491F1eFBeEc0261eb556f0973C7
Name	AimBot
Token Tracker	AimBot (AIMBOT)
Decimals	18
Supply	1,000,000
Platform	ETHEREUM
compiler	v0.8.19+commit.7dd6d404
Contract Name	AimBot
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://etherscan.io/ token/0x0c48250eb1f29491f1efbeec0261eb556f0973c7#code
Payment Tx	Corporate

Main Contract Assessed
Contract Name

Name	Contract	Live
AimBot	0x0c48250Eb1f29491F1eFBeEc0261eb556f0973C7	Yes

TestNet Contract Assessed
Contract Name

Name	Contract	Live
AimBot	0x8D2080302F552c07ddab3fFE890547941420E02F	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
AimBot	ad991903a9e69c123a8ef32016ba0fdf80d2548b	AimBot.sol
AimBot	f18813fa5dcbbfe87bcc8d38f7945510e3336b82	AimBotDividends.sol
AimBot	2584c945324f7fbecfe65c2874d6b8b01f04cc67	Ownable.sol
AimBot	7288ffa106491bc0bb70b7522e1cfa24b4ac1bee	Context.sol
AimBot	f11fb6e097c005c4d69bea282436d5570af586da	ERC20.sol
AimBot	undefined	IERC20.sol

Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	AimBot.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	AimBot.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	AimBot.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	AimBot.sol	L: 16 C: 0
SWC-104	Pass	Unchecked Call Return Value.	AimBot.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	AimBot.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	AimBot.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	AimBot.sol	L: 0 C: 0
SWC-108	Low	State variable visibility is not set..	AimBot.sol	L: 21 C: 33
SWC-109	Pass	Uninitialized Storage Pointer.	AimBot.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	AimBot.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	AimBot.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	AimBot.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	AimBot.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	AimBot.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-115	Low	Authorization through tx.origin.	AimBot.sol	L: 142 C: 3810
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	AimBot.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	AimBot.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	AimBot.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	AimBot.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	AimBot.sol	L: 122 C: 27
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	AimBot.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	AimBot.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	AimBot.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	AimBot.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	AimBot.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	AimBot.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	AimBot.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	AimBot.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	AimBot.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	AimBot.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	AimBot.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	AimBot.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	AimBot.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	AimBot.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	AimBot.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	AimBot.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.

Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.

Smart Contract Vulnerability Details

SWC-108 - State Variable Default Visibility

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables

Smart Contract Vulnerability Details

SWC-115 - Authorization through tx.origin

CWE-477: Use of Obsolete Function

Description:

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

Remediation:

tx.origin should not be used for authorization. Use msg.sender instead.

References:

Solidity Documentation - tx.origin

Ethereum Smart Contract Best Practices - Avoid using tx.origin

SigmaPrime - Visibility.

Smart Contract Vulnerability Details

SWC-120 - Weak Sources of Randomness from Chain Attributes

CWE-330: Use of Insufficiently Random Values

Description:

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Shadowing state variables can also occur within a single contract when there are multiple definitions on the contract and function level.

Remediation:

Using commitment scheme, e.g. RANDAO. Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles. Using Bitcoin block hashes, as they are more expensive to mine.

References:

How can I securely generate a random number in my smart contract?)

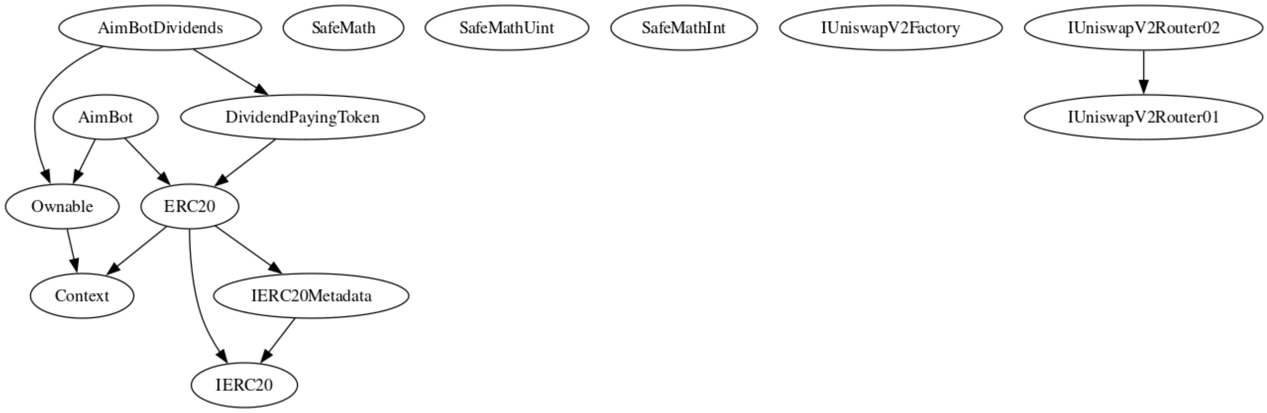
When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.

Inheritance

The contract for AimBot has the following inheritance structure.

The Project has a Total Supply of 1,000,000



Privileged Functions (onlyOwner)

Please Note if the contract is Renounced none of this functions can be executed.

Function Name	Parameters	Visibility
updateBotWallet	address _botWallet	External
updateDividends	address _dividends	External
updateFee	uint256 _totalFee, uint256 _botFee	External
updateMaxHoldingPercent	uint256 percent	Public
updateSwapAt	uint256 value	External
openTrading		External

AIMBOT-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	🟢 Low	AimBot.sol: L: 68 c:14, L:72 C14	Detected

Description

The given input is missing the check for the non-zero address.

The given input is missing the check for the missing required function.

Remediation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
  ...
   require(receiver != address(0), "Receiver is the zero address");
  ...
  ...
  require(value X limitation, "Your not able to do this function");
 ...
```

We also recommend customer to review the following function that is missing a required validation. missing required function.

AIMBOT-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	🟢 Low	AimBot.sol: L: 68 c:14, L:72 C:14,L: 82 C:14,L: 88 C:14,L: 93 C:14,L: 106 C:14	Detected

Description

Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.The linked code does not create an event for the transfer.

Remediation

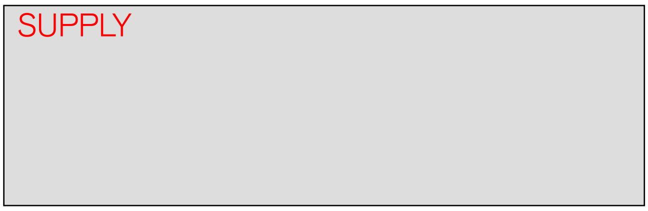
Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

AIMBOT-07 | State Variables could be Declared Constant.

Category	Severity	Location	Status
Coding Style	● Low	AimBot.sol: L: 25 C: 14	Detected

Description

Constant state variables should be declared constant to save gas.



```
SUPPLY
```

Remediation

Add the constant attribute to state variables that never changes.

https://docs.soliditylang.org/en/latest/contracts.html#constant-state-variables

AIMBOT-08 | Dead Code Elimination.

Category	Severity	Location	Status
Coding Style	🟢 Low	AimBot.sol: L: 38 C:14	Detected

Description

Functions that are not used in the contract, and make the code s size bigger.

```
receiveBlock
```

Remediation

Remove unused functions. dead-code elimination (also known as DCE, dead-code removal, dead-code stripping, or dead-code strip) is a compiler optimization to remove code which does not affect the program results. Removing such code has several benefits: it shrinks program size, an important consideration in some contexts, and it allows the running program to avoid executing irrelevant operations, which reduces its running time. It can also enable further optimizations by simplifying program structure.

https://docs.soliditylang.org/en/latest/cheatsheet.html

AIMBOT-13 | Extra Gas Cost For User.

Category	Severity	Location	Status
Logical Issue	ⓘ Informational	AimBot.sol: L: 155 C:14	Detected

Description

The user may trigger a tax distribution during the transfer process, which will cost a lot of gas and it is unfair to let
a single user bear it.

Remediation

We advise the client to make the owner responsible for the gas costs of the tax distribution.

Project Action

AIMBOT-20 | This loads the contract with tokens..

Category	Severity	Location	Status
Logical	🔴 Critical	AimBot.sol: L: 44 C: 14	Detected

Description

The contract is designed to preload a specified quantity of tokens for exchange during the swap process. This action has the potential to exert downward pressure on the token's value, which could result in a market sell-off. It is important to carefully consider the implications of this feature to mitigate any adverse effects on the project's economy.

Remediation

Ensure protection against significant asset liquidation events.

Project Action

Technical Findings Summary

Classification of Risk

Severity	Description
🔴 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟠 High	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟡 Medium	Risks may not pose a direct risk to users’ funds, but they can affect the overall functioning of a platform
🟢 Low	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
ℹ Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
🔴 Critical	1	0	1
🟠 High	0	0	0
🟡 Medium	0	0	0
🟢 Low	4	4	0
ℹ Informational	1	1	0
Total	6	5	1

Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/aimbot_ai	Pass
Other		Fail
Website	https://aim-bot.app/	Pass
Telegram	https://t.me/Aimbotportal	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:

Assessment Results

Score Results

Review	Score
Overall Score	79/100
Auditor Score	85/100
Review by Section	Score
Manual Scan Score	26
SWC Scan Score	29
Advance Check Score	24

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project most pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below.

Audit Fail

Assessment Results

Important Notes:

- Ownership Concentration: The constructor mints a significant portion of the supply to the contract and the owner, which could pose a risk of centralization.ı

- Max Wallet Size: The maxWallet variable limits the amount of tokens one wallet can hold, which can prevent whale manipulation but needs to be communicated clearly to users.ı

- Trading Restrictions: The openTrading function sets initial trading parameters, including a max wallet size of 1% of the total supply. This could be a point of contention if not made clear to users.ı

- Anti-sniping Mechanism: The contract imposes a high snipeFee for the first few blocks after trading starts to deter bots, but this could also affect regular users who buy in early.ı

- Fee Structure: The updateFee function allows the owner to change fees, but it's capped at 5% for totalFee and botFee must be less than or equal to totalFee.ı

- Dividend Exclusions: The contract has functions to exclude addresses from dividends, which is a common feature but should be transparent to users.ı

- Swap Mechanism: The contract can swap tokens for ETH and distribute funds, which could be a point of failure if not properly secured. The swapTokensForEth function is private and can only be called internally.ı

- Contract Interaction Checks: The isContract function is used to prevent contracts from buying in the first few blocks, but this might not be foolproof against sophisticated bots.

Auditor Score =85
Audit Fail

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that actagainst the nature of decentralization, such as explicit ownership or specialized access roles incombination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocation for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access 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 with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

ASSURE DEFI™
THE VERIFICATION GOLD STANDARD