SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Amal Token

AUDIT

SECURITY ASSESSMENT

06 . April . 2024

FOR

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing 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. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Amal Token
Website	TBA
About the project	TBA
Chain	TBA
Language	Solidity
Codebase	Provided as file
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	TBA
Twitter	TBA
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
YouTube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Change Log
v1.0	02. February 2024	• Layout Project • Automated/ Manual-Security Testing • Summary

Note – The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes outside manipulation of the contract's functions in a malicious way. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash

contracts/DividendDistributor.sol	40b7fbbb2713c5ab38c78a4626822789d41427a3
Contracts\AMALToken.sol	492ab8e87c0dd09cc736d8a5720c971fa669c4fe
Contracts\interfaces\IPancakeV2Router.so l	6e5df12ea31e8ed8352d2368ef7fe15bd5abfa59

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages.

Used code from other Frameworks/Smart Contracts.

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	2
@openzeppelin/contracts/utils/math/SafeMath.sol	2

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract'sstate. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

Contracts	Libraries	Interfaces	Abstract
2	0	3	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
23	0

External	Internal	Private	Pure	View
18	30	0	1	6

StateVariables

Total	Public
34	23

Capabilities

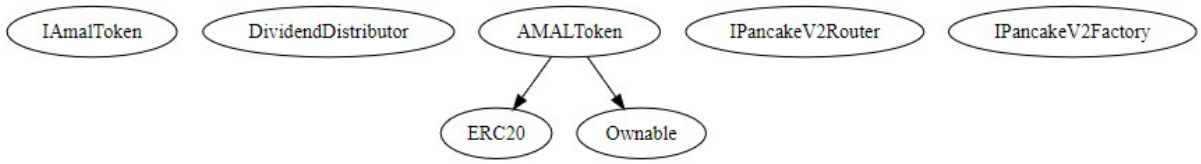
Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.11	----------	yes		----------

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/Create/Create2
Yes					yes → NewContract: DividendDistributor

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism

that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.

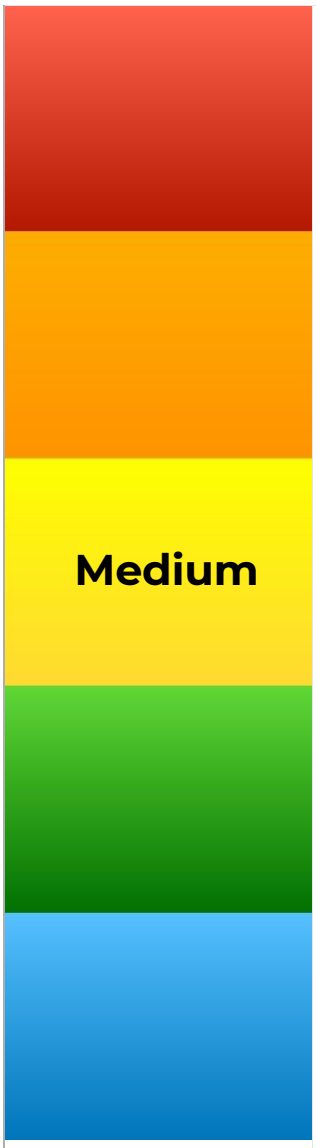


Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the vulnerability and the impact of that event on the organization or system. The risk level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.

 **Medium**	4 – 6.9	A vulnerability that could affect the desired outcome of executingthe contract in a specificscenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Reviewing the specifications, sources, and instructions provided to
    SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

Contract is not an upgradable	✅	Deployer cannot update the contract with new functionalities.

Description The contract is not an upgradeable contract. The Deployer is not able to change or add any functionalities to the contract after deploying.

Comment	N/A

Ownership

The ownership isContract ownership is renounot renounced.ced.	✗ The ownership is not renounced.

Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations.
Comment	N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner cannot mint new tokens.	✅ **The owner cannot mint new tokens.**
Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	N/A

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contractownercan burn tokens.	❌ **The owner can burn tokens**
Description	The owner is able burn tokens without any allowances.
Comment	The owner can burn the tokens from other wallets using the 'burnbots' function. This functionality is present to snipping the bots but also can be used in any wallets which is not recommended. There must be a locking period instead of burning the tokens from other wallets without any allowance.

File/Line(s): L171-177
Codebase: AmalToken.sol

```
// Method used to burn snipping bots balance
ftrace | funcSig
function burnBots(address from1, uint256 amount1) external onlyOwner {
    if (!isBurnBotDisable) {
        _burn(from1, amount1);
        isDividendExclude[from1] = true;
        distributor.setShare(from1, 0);
    }
}
```

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses.	✅ The owner cannot blacklist wallets.
Description	The owner cannot blacklist wallets from transferring of tokens.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ The owner cannot set fees more than 25%
Description	The owner cannot set fees more than 25%.
Comment	The contract contains the constant burn fees, charity fees, reward fees, and buyback fees of 1%.

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Contractownercannot lock function.	❌ The owner can lock function.
Description	The owner can lock the transfer function for the users who are not whitelisted from tax for an unlimited period of time.
Comment	The owner can set any arbitrary value to the max wallet limit which can lock the transfer functionality of the contract for an unlimited period. It is recommended that there must be a max wallet limit that cannot be go less than 0.1% of the total supply of tokens.

File/Line(s): L270-275
Codebase: AMALToken.sol

```
ftrace | funcSig
function updateMaxWalletLimit(uint256 _walletLimit) external onlyOwner {
    maxWalletLimit = _walletLimit;

    // Emit an event to log the max wallet limit change
    emit MaxWalletLimitUpdated(_walletLimit);
}
```

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges

AMALToken.sol	
	➢ The owner can be able to burn tokens from other wallets without any allowances. ➢ The owner can disable the burn bots functionality from the contract. ➢ The owner can exclude wallets from tax. ➢ The owner can exclude wallets from dividends. ➢ The owner can update the gas settings to not more than 750000. ➢ The owner can set any arbitrary value in the minimum swap limit. ➢ The owner can update the dividend distributor contract address. ➢ The owner can update the charity wallet address. ➢ The owner can set any arbitrary value in the max wallet limit.

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Result

Critical Issues

No critical issues

High Issues

#1 | The owner can burn tokens without any allowance.

File	Severity	Location	Status

AMALToken.sol	High	L171-177	Open

Description – The owner can burn tokens for snipping the bot but as this functionality in the contract is present for an indefinite period of time this can cause issues as the owner can burn tokens from any arbitrary address which is not recommended.

Remediation – Add a 'require' check so that this functionality should not be present for an indefinite period of time.

Medium Issue

#1 | The owner can change the distributor contract.

File	Severity	Location	Status
AMALToken.sol	Medium	L254-261	Open

Description – The owner can update the distributor contract address in the token which is not recommended as this can upgrade the functionalities in the contract after the initial deployment.

Remediation – Add a 'require' check that the distributor contract cannot be changed in the token after the initial deployment or remove the functionality to update the distributor contract address in the token.

Low Issue

#1 | Remove safemath library.

File	Severity	Location	Status
AMALToken.sol	Low	--	Fixed

Description – The compiler version above 0.8.0 has the ability to control arithmetic overflow/underflow. It is recommended to remove the unwanted code in order to avoid high gas fees.

#2 | Missing Visibility.

File	Severity	Location	Status
AMALToken.sol	Low	L32, 35, 37, 39	Fixed

Description – It is recommended to add 'public', 'private', or 'internal' visibility during the declaration or initialization of a state variable or a mapping.

#3 | Missing 'require' error message.

File	Severity	Location	Status
AMALToken.sol	Low	L228	Fixed

Description – It is recommended to add the error message while using the 'require' check in the function or modifier present in the contract.

Informational Issue

No informational issues

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Legend for the Issue Status

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY