SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

Unamano

AUDIT

SECURITY ASSESSMENT

16. October, 2024

FOR

unamano

SolidProof_io

@solidproof_io

Introduction

SolidProof.io is a brand of the officially registered company Future Visions Deutschland. We're mainly focused on Blockchain Security, such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assesses potential security issues in the smart contracts implementations, reviews for potential inconsistencies between the code base and the whitepaper/documentation, and provides suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should they be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should they be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io does not cover testing or auditing the integration with external contracts 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 analysed, 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 analyse.

Project Overview

Summary

Project Name	Unamano
Website	https://unamano.io/
About the project	Unamano has evolved into a cross-chain POS liquid mining protocol supporting both Ethereum (ETH 2.0) and Binance Smart Chain (BSC). Through its upgraded structure, users can stake ETH and other tokens to earn multiple crypto assets from a wider ecosystem.
Chain	BSC
Language	Solidity
Codebase Link	https://bscscan.com/address/0x17C9072D7639616507D74423E3F7EB8C1B6075F8#code
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/unamanoofficial
Twitter	https://x.com/unamanoio
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	https://medium.com/@unamanoio
Discord	https://discord.com/invite/eMXk9m9Kpt
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	16. October 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 malicious outside manipulation of the contract's functions. 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
StarterETH.sol	cbc125d61c1284def07920f6817a2fa730a72355
Include.sol	26a7ea8ce49b75ddb13a2f9e7169b872b2d2e275

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) indicate a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

N/A

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

Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit 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 executing the contract in a specific scenario.	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. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
   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 is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., 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 an upgradeable	❌ Deployer can update the contract with new functionalities
Description	The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	The deployer can deploy a new version of the contract after the initial deployment of the contract.

Ownership

The ownership is not renounced	❌ The owner is not renounce
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
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

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 refers 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 can 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 without Allowance

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.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens**
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

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 addresses**
Description	The owner is not able blacklist addresses to lock funds.
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 contract's cost, 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 levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	N/A

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 tokens 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.

Contract owner can lock the user funds	❌ **The owner is able to lock the contract**
Description	Locking the contract means that the owner is able to lock any funds of addresses that they are not able to transfer bought tokens anymore.
Example	An example of locking is by pausing the contract or blacklisting any addresses. That causes that the blacklisted address is not able to transfer (buy/sell) anymore.
Comment	The contract contains the functionality in which the governance role can claim the tokens from the contract this can lead to the locking of claim function if there are no tokens in the contract the user will not be able to claim the tokens from the contract. There must be a check so that the governance or any other cannot rain the staked tokens from the contract.

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's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
20	5	3	5

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
85	16

External	Internal	Private	Pure	View
45	149	2	14	40

StateVariables

Total	🌐Public
52	29

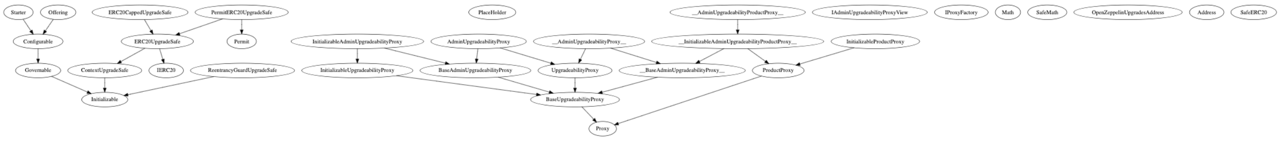
Capabilities

Solidity Versions observed	Transfers ETH	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.6.0`	ABIEncoderV2	yes	yes (16 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
yes		yes	yes	yes	

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.


Starter
Offering
ERC20CappedUpgradeSafe
PermitERC20UpgradeSafe
PlaceHolder
__AdminUpgradeabilityProductProxy__
IAdminUpgradeabilityProxyView
IProxyFactory
Math
SafeMath
OpenZeppelinUpgradesAddress
Address
SafeERC20
Configurable
ERC20UpgradeSafe
Permit
InitializableAdminUpgradeabilityProxy
AdminUpgradeabilityProxy
__AdminUpgradeabilityProxy__
__InitializableAdminUpgradeabilityProductProxy__
InitializableProductProxy
Governable
ContextUpgradeSafe
IERC20
ReentrancyGuardUpgradeSafe
InitializableUpgradeabilityProxy
BaseAdminUpgradeabilityProxy
UpgradeabilityProxy
__BaseAdminUpgradeabilityProxy__
ProductProxy
Initializable
BaseUpgradeabilityProxy
Proxy

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 a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
StarterETH.sol	• The governance role can initialize the currency, underlying, price, time and time settle value in the contract. • The governance role can withdraw the tokens to the recipient wallet. • The governance role can withdraw the tokens from the contract to the wallet including the current and underlying tokens. • The governance role can withdraw the ETH from the contract. • The governance role can set quota for the wallets. • The governance role can claim tokens and ETH from the contract.
Include.sol	• The governance role can set config in the contract.

Recommendations

To avoid potential hacking risks, the client should 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 can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

#1 I Missing access control.

File	Severity	Location	Status
Include.sol	High	L42-61	Open

Description - The contract contain the functionality in which anyone can call the delegate function of the proxy contract which is not recommended as this can modify and upgrade the current version of the contract. There must be a check so that the selected user should only be able to call this function from the contract.

Medium issues

#1 I Claim function can be locked

File	Severity	Location	Status
StarterETH.sol	Medium	L277-283	Open

Description - The contract contains the functionality in which the governance role can claim the tokens from the contract this can lead to the locking of claim function if there are no tokens in the contract the user will not be able to claim the tokens from the contract. There must be a check so that the governance or any other cannot rain the staked tokens from the contract.

Low issues

#1 I Missing Zero Address Validation

File	Severity	Location	Status
StarterETH.sol	Low	L31-38	Open

Description - Make sure to validate that the address passed in the function parameters is “non-zero”.

#2 I Missing Events

File	Severity	Location	Status
StarterETH.sol	Low	L31-38	Open

Description - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

#3 I Old Compiler version

File	Severity	Location	Status
All	Low	—	Open

Description - The contracts use outdated compiler versions, which are not recommended for deployment as they may be susceptible to known vulnerabilities.

Remediation - Use a newer pragma version. At least use the 0.8.20 version.

#4 I Unnecessary code.

File	Severity	Location	Status
StarterETH.sol	Low	L302-304	Open

Description - The contract uses both functions trigger the settle() function when no Ether is sent, having both might be redundant. The receive() function is more specific for handling Ether transfers, so you could potentially remove the fallback() function if it's not necessary to have two separate entry points.

Informational issues

#1 I NatSpec documentation missing

File	Severity	Location	Status
Main	Informational		Open

Description - If you started to comment on your code, comment on all other functions, variables etc.

#2 I Contract doesn't import npm packages from source (like OpenZeppelin etc.)

File	Severity	Location	Status
All	Informational	N/A	Open

Description - We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or can be susceptible to vulnerabilities.

Legend for the Issue Status

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.

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY