ChainAudits

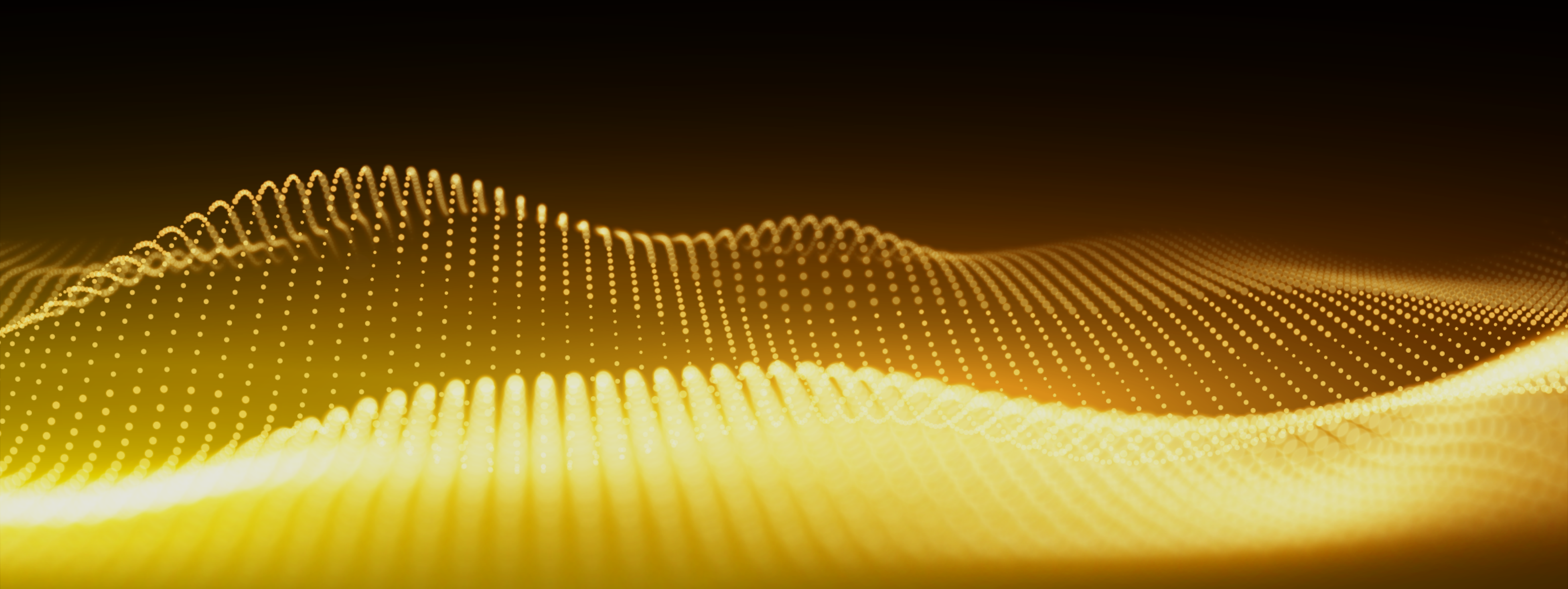
Smart Contract Audit

Security Assessment

29.08.2024

ProBit Global

PROBIT
GLOBAL



Project Details

CODEBASE Deployed	NETWORK ETH	LANGUAGE Solidity
NAME ProBit Global	TOTAL SUPPLY Customizable	SYMBOL PROB
WEBSITE Active	UNIT TESTS Not Provided	FORK N/A

ABOUT THE PROJECT

ProBit Global is a Top 20 crypto exchange worldwide providing unlimited access to trade and buy Bitcoin, Ethereum and 600+ altcoins in 1000+ markets.

WEBSITE	DEPLOYED CONTRACT
https://www.probit.com/en-us/	0xfB559CE67Ff522ec0b9Ba7f5dC9dc7EF6c139803

Social Media Links

Telegram	https://t.me/ProBitGlobalOfficial
Twitter/X	https://x.com/ProBit_Exchange/
Facebook	https://www.facebook.com/probitexchange/
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	https://probit-exchange.medium.com/
Discord	https://discord.com/invite/uK7hayUHxu
Youtube	N/A
TikTok	N/A
LinkedIn	https://www.linkedin.com/company/probit-global/

Version	Delivery Date	Changelog
v1.0	29. August 2024	• Layout Project • Automated- /Manual-Security Testing • Summary

Vulnerability Summary

● Critical	Critical risks are those that affect the platform's safe operation and must be resolved before launch. Users should avoid investing in any project with unresolved critical risks.
● Major	Major risks include centralization issues and logical errors. These risks can potentially result in the loss of funds or control over the project under certain conditions.
● Medium	Medium risks might not directly threaten users' funds, but they can impact the platform's overall functionality.
● Minor	Minor risks are similar to the above categories but on a smaller scale. They typically do not compromise the project's overall integrity but may lead to less efficient solutions.
● Informational	Informational errors are recommendations aimed at improving code style or aligning operations with industry best practices. These usually do not affect the code's overall functionality.

Note – The provided audit report thoroughly examines the security aspects of the smart contract employed in the project, encompassing potential malicious manipulation of the contract's functions from external sources. However, it's important to note that this analysis does not incorporate functional or unit testing of the contract's logic. Therefore, we cannot ensure the absolute correctness of the contract's logic, including internal calculations within the formulae utilised in the contract.

Overview

In-Scope Files

The team provided us with files to review during security audits. This audit covered the following files listed below with their respective SHA-1 Hash.

SHA-1	File
e21363a8cf0768807c5a9f7e73fda88ae5c306c3	PROB.sol

Please note that files with hash values different from those listed in this table have been changed after security checks, intentionally or unintentionally, as a particular hash value may indicate a changed state or potential vulnerabilities not checked in this scan.

Note for Investors: We only examined agreements indicated in the indicated ratings. No contracts associated with the project beyond this range have been audited, therefore, we cannot provide insight or assume responsibility for their security.

Imported packages

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

N/A

External/Public functions

External/public functions can be invoked outside of the contract, i.e., accessed by other contracts or external accounts on the blockchain. These functions are identified using external or public visibility modifiers in the function declaration.

State variables

State variables are stored on the blockchain as part of the contract conditions. They are declared at the contract level and can be accessed and changed by any action in the contract (except with modifiers like onlyOwner, etc.). State transitions can be described using a visibility modifier, such as public, private, or internal, which refers to the access to the transition.

Components

Contracts	10
Libraries	0
Interfaces	0
Abstract	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	30
Payable	2
External	0
Internal	36
Private	0
Pure	12
View	8

State Variables

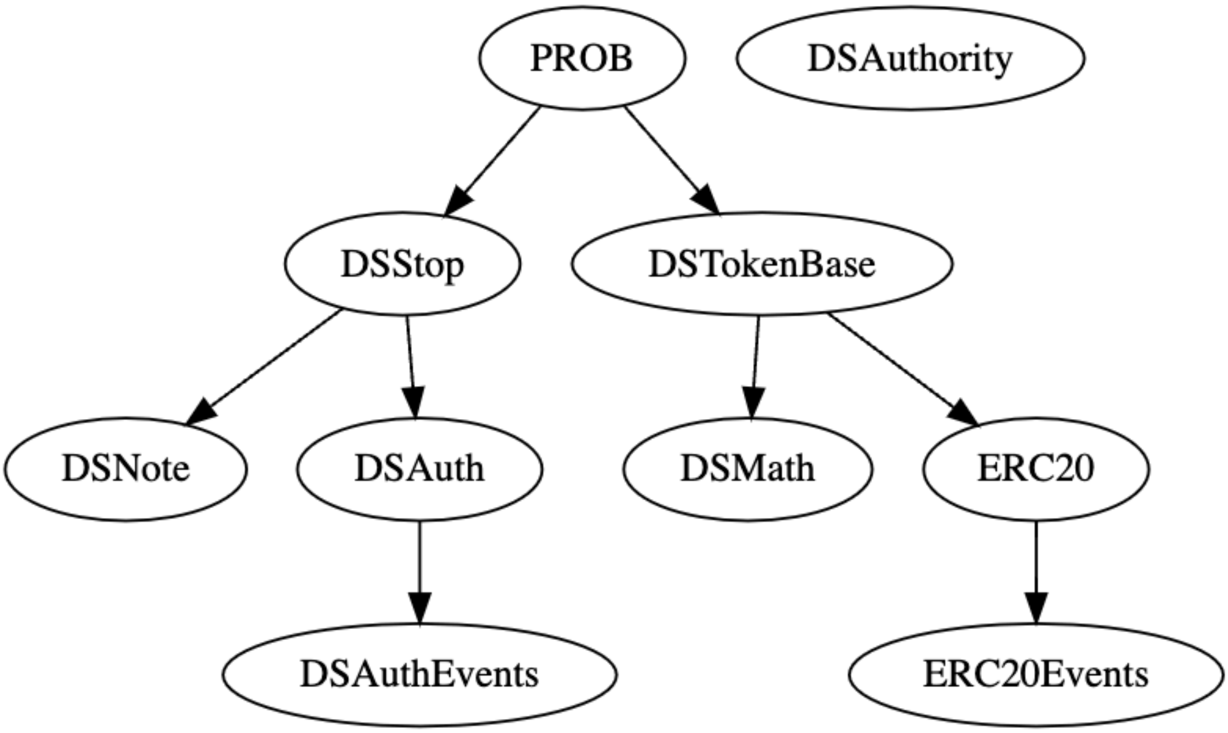
Total	11
Public	6

Capabilities

Solidity Versions observed	^0.4.25
Transfers ETH	No
Can Receive Funds	No
Uses Assembly	No
Has Destroyable Contracts	No

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

Strategies

While our projects undergo rigorous testing to address all known vulnerabilities, it's important to provide further clarification and understanding by outlining specific vulnerabilities that have been identified. This additional information enhances transparency and helps stakeholders comprehend the security measures taken and any potential risks mitigated.

Title	Issue
Re-entrancy	Improper Enforcement of Behavioral workflow
Unexpected Ether Balance	Improper Locking
Code With No Effects	Irrelevant Code
Flash Loan Attacks	External Oracle Manipulation
Sandwich Attacks	A Form of Front Running
Write to Arbitrary Storage Location	Improper Write to Arbitrary Storage Location
Variable Shadowing	Improper Coding Standards
Unprotected SELF-DESTRUCT	Improper Access Control
Potential Honeypot	Improper Function
Unprotected ETH withdrawal	Improper Access Control
Outdated Compiler Version	Using components with Known vulnerabilities
Weak Sources of Randomness from Chain Attributes	Use of Insufficiently Random values
Unsafe use of libraries	Improper Implementation
Wrong Implementation of Token Standards	Improper Coding Standards

Auditing Strategy and Techniques Applied

All Projects at ChainAudits undergo our in-house developed "4-Eye" method. This process ensures that the code is analyzed not by a single auditor but by our entire technical team. Moreover, this is accomplished most efficiently, leaving no stone unturned.

We manually check every file, line by line. Automated tools are used solely to help us achieve faster and better results.

Code Analysis Methodology

The auditing process follows a routine series of steps:

- Manual Code Review:
  - Evaluate the overall structure and organization of the smart contract code line-by-line.
  - Review the implementation of access control mechanisms to ensure that sensitive functions are appropriately restricted to authorized users.
  - Ensure that critical vulnerabilities are adequately tested and that edge cases and boundary conditions are covered.
- Static Code Analysis:
  - Utilise static analysis tools to scan the codebase for vulnerabilities and security issues.
  - Identify common vulnerabilities like reentrancy, integer overflow/underflow, and unchecked external calls.
  - Evaluate compliance with coding standards and best practices, ensuring adherence to security guidelines.
- Code Structure and Architecture Review:
  - Analyze the overall structure and architecture of the smart contract code.
  - Assess the modularity, readability, and maintainability of the code.
  - Review the separation of concerns and adherence to design patterns for robustness and security.
- Security Best Practices Evaluation:
  - Evaluate the implementation of security best practices, including access control mechanisms and input validation.
  - Verify proper error handling to mitigate potential vulnerabilities and ensure contract robustness.

  - Check for gas optimization techniques to enhance efficiency and reduce transaction costs.
- External Dependency Assessment:
  - Assess the integration with external contracts, libraries, or services.
  - Review the security of external dependencies and their impact on the overall security of the smart contract codebase.
  - Ensure secure interactions with external components to prevent vulnerabilities and attack vectors.
- Post-Analysis Support:
  - Offer support and guidance to assist the development team in addressing identified vulnerabilities.
  - Collaborate with stakeholders to ensure effective implementation of recommended security enhancements.
  - Provide ongoing assistance and consultation to maintain the security of the smart contract codebase.

Discovered Issues

Privileges

Centralization emerges when one or multiple entities possess privileged access or authority over a smart contract's functionalities, data, or decision-making processes. This situation may arise if a singular entity exercises complete control over the contract or if certain participants hold unique permissions or capabilities inaccessible to others within the ecosystem.

In the project, some authorities have no access to any functions as the ownership has been renounced:

No.:	File	Privileges
1	PROB	• Minting of tokens without any limits • Pause/Unpause trading in the cotnract

Recommendations

To mitigate potential hacking risks, the team must meticulously handle the private key associated with the privileged account. Moreover, we advise bolstering the security measures surrounding centralized privileges or roles within the protocol by implementing decentralized mechanisms or employing smart-contract-based accounts like multi-signature wallets. This approach can enhance security by distributing control among multiple parties and requiring consensus for executing critical actions, thereby reducing the likelihood of unauthorized access or malicious activity.

Here are some suggestions of what the project owners can do:

- **Adopting multi-signature wallets**: Utilize wallets like Gnosis Safe that mandate approval from multiple parties before executing transactions. This additional layer of security helps safeguard against unauthorized actions.
- **Implementing a timelock**: Introduce a timelock feature with a delay of, say, 48-72 hours for sensitive operations. This delay period ensures stakeholders have ample time to review and respond to proposed changes, reducing the risk of impulsive or unauthorized modifications.
- **Incorporating a DAO/Governance/Voting module**: Integrating a decentralized governance system enhances transparency and community involvement. This empowers users to participate in decision-making processes, fostering a sense of ownership and accountability within the community.
- **Renouncing ownership**: Consider relinquishing ownership rights once all necessary configurations are in place. By doing so, the owner forfeits the ability to alter contract variables, enhancing trust and decentralization. It's crucial to ensure all settings are finalized before renouncing ownership to prevent unintended consequences.
- These measures collectively strengthen the security and integrity of the smart contract, mitigating risks and promoting a more resilient and transparent ecosystem.

Upgradeability

Deployer cannot update the contract with new functionalities

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

Ownership

Issue-ID	Severity	Location	Status
PRO–01	● Major	N/A	Pending

Description

Ownership is not renounced, and since the contract allows the owner to mint unlimited tokens and prevent users from transferring their tokens, we highly recommend removing ownership from the contract.

The owner is not renounced and can modify the contract

The owner is able to change the state variables. The contract remains under the control of the owner. This means that the owner can change the state variables at any time and prevent transactions if necessary.

Note – If the contract remains undeployed, we would regard the ownership as not relinquished. Additionally, without any ownership functionalities, the ownership is presumed to be automatically renounced.

Authorities Permissions

Functions that alter the state and are equipped with access control can pose risks. Recognizing that misuse of these functionalities can result in financial losses is essential. We offer a guide to understand the implications of presence/absence functions further.

Minting tokens

Minting involves creating new tokens within a cryptocurrency or blockchain network. This task is commonly undertaken by the project's owner or an assigned authority, granting them the capability to increase the network's total token supply.

Issue-ID	Severity	Location	Status
PRO–02	● Critical	L271	Pending

Description

The owner can mint unlimited tokens without any conditions or limits. This means that the owner can mint tokens anytime and may abuse this power to manipulate the price of the tokens.

Recommendation

We recommend renouncing the contract as we cannot recommend updating the code since it is impossible now due to the fact that the project has been running for a long time and there is no migration system implemented in the code.

Authorities can mint new tokens

The ability for owners to mint new tokens allows them to reward themselves or others involved in the project, with the potential to trade these tokens for capital on cryptocurrency exchanges. However, this power comes with the risk of misuse, which could erode trust and confidence in the project. If stakeholders perceive that the owner is unfairly or secretly minting tokens, it could lead to decreased demand and a significant drop in the token’s value, undermining the entire platform.

Burning Tokens

Burning tokens involves permanently removing a specific quantity of tokens from circulation, decreasing the total supply of a cryptocurrency or token. This practice is typically undertaken to enhance the value of the remaining tokens. By reducing the overall supply, burning creates a sense of scarcity, potentially stimulating demand and driving up the token's value.

Authorities cannot burn tokens without approval

Authorities are not able to burn tokens without any allowances. Only the addresses with allowances can burn tokens from other wallets

Note – In the context of ProBit contract, the tokens can only be burned by their rigthful owners. Meaning that the users and owners both can only burn the tokens that they own.

Blacklist addresses

Blacklisting addresses in smart contracts involves adding specific addresses to a blacklist, thereby preventing them from accessing or participating in certain functionalities or transactions within the contract. This measure is useful for preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Authorities cannot prevent function calls

Authorities are not able to prevent addresses from buying/selling.

Fees and Tax

In specific smart contracts, the individual or entity responsible for creating the contract retains the ability to establish fees for specific actions or functionalities within it. These fees serve various purposes, including covering operational expenses such as gas fees or compensating the contract's creator for their contributions in developing and managing the contract.

> **Authorities cannot update fees**
>
> Authorities are not able to update the fees for buying/selling.

Lock User Funds

In the context of a smart contract, locking entails confining access to specific tokens or assets for a predetermined duration. While tokens or assets are locked within the smart contract, they become inaccessible for transfer or utilization until the conclusion of the lock-up period or the fulfillment of specific conditions stipulated within the contract.

Issue-ID	Severity	Location	Status
PRO–03	● Critical	L238, 234	Pending

Authorities can lock funds

Authorities can lock funds.

Description

The owner can pause/unpause the "transferFrom" and the "approve" functions, which will result in the user funds being locked and un-transferrable, which will transform the smart contract into a HoneyPot.

Recommendation

We recommend to renounce the ownership.

NatSpec Documentation missing

Issue-ID	Severity	Location	Status
PRO–04	● Informational	N/A	Pending

Description

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

The contract doesn't import npm packages from the source (like OpenZeppelin, etc.)

Issue-ID	Severity	Location	Status
PRO–05	● Informational	N/A	Pending

Description

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

Missing events

Issue-ID	Severity	Location	Status
PRO–06	● Informational	L286	Pending

Description

Ensure that events are emitted for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

Old compiler version

Issue-ID	Severity	Location	Status
PRO–07	● Minor	L05	Pending

Description

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

Recommendation

Use the latest pragma version, or at least use the 0.8.18 version.

Function Overloading and Potential Confusion in approve Functionality

Issue-ID	Severity	Location	Status
PRO–08	● Minor	L05	Pending

Description

In the PROB contract, there are two overloaded approve functions: one with a single address parameter and another with both address and uint parameters. Overloading can lead to potential confusion because the single-parameter approve function is not part of the ERC20 standard. Moreover, calling super. approve from the single-parameter function triggers the two-parameter version in the parent contract, which may lead to unexpected behaviour or errors. It's recommended to avoid such overloading to ensure compliance with ERC20 standards and maintain clarity in the contract's functionality.

Recommendation

1. **Remove the Single-Parameter approve Function:** Eliminate the overloaded approve function that only takes an address as a parameter. This will ensure that only the standard ERC20 approve function (which takes both an address and a uint amount) is present.

2. **Standardize approve Functionality:** Ensure that the approve function strictly adheres to the ERC20 standard by accepting both the spender address and amount as parameters.

This will improve code clarity, prevent unexpected behaviour, and maintain compliance with the ERC20 standard.

Disclaimer

ChainAudits reports should not be construed as an endorsement or disapproval of any particular business or group. These reports do not reflect the economic value of any of the products or assets developed by the group. Also, ChainAudits does not consider integration with external contracts or services (e.g., Unicrypt, Uniswap, PancakeSwap).

ChainAudits reports aim to identify successful audit processes to help our clients improve the quality of the code and manage the risks associated with cryptographic tokens and blockchain technology. It is important to understand that blockchain technology and cryptographic assets pose significant ongoing risks. Each company and individual should conduct its own due diligence to maintain consistent safety measures. ChainAudits makes no representations about the security or performance of the technologies we audit.

ChainAudits does not provide any warranty or guarantee that the analyzed technology is completely defect-free, nor does it imply approval by the technology's owners. These audits should not be used to make input or output decisions; they will be involved in any project. They are not giving financial advice and should not be construed as such.