BLOCKAPEX

CANISTER SECURITY

V 1.0

DATE: 8th AUG 2024

PREPARED FOR: META POOL

SECURITY REPORT
BLOCKAPEX VERIFIED

About BlockApex

Founded in early 2021, is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

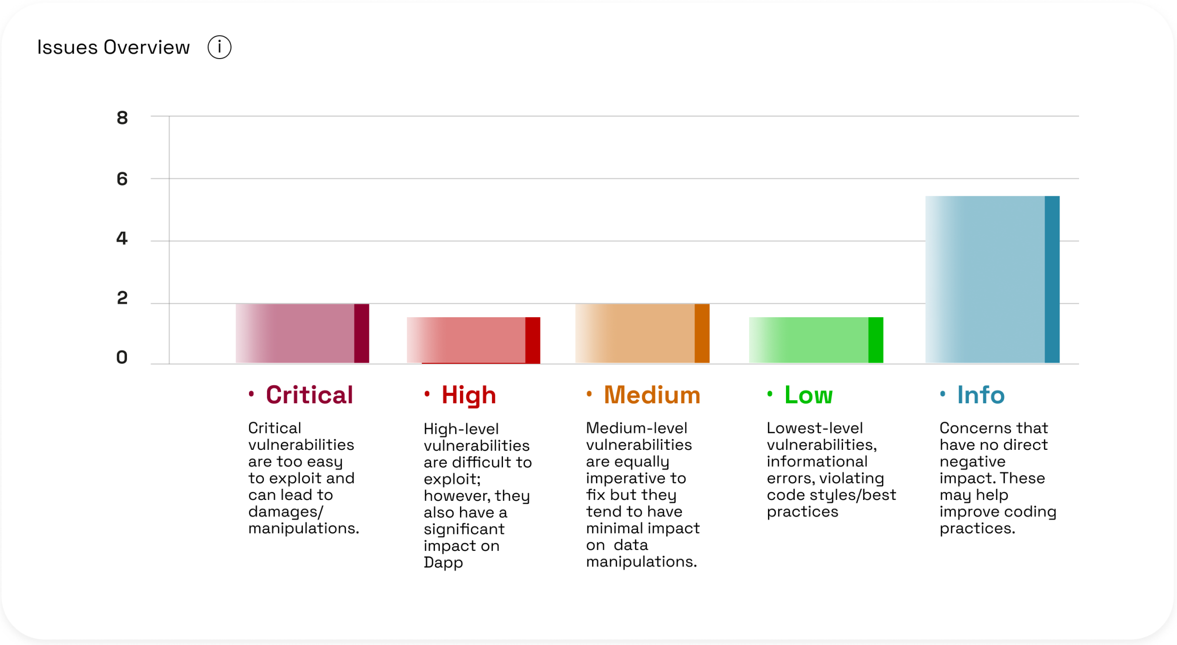
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

1 Executive Summary

Our team performed a technique called Filtered Audit, where two individuals separately audited the StakedICP Canisters. After a thorough and rigorous manual testing process involving line by line code review for bugs. All the raised flags were manually reviewed and re-tested to identify any false positives.



1.1 Project Overview

StakedICP is a liquid staking solution for ICP (Internet Computer Protocol) tokens. It allows users to stake their ICP without the need to lock their assets or maintain the staking infrastructure. Instead, users can participate in on-chain activities such as lending while earning staking rewards.

When users stake their ICP through StakedICP, they receive stICP tokens that represent their staked ICP and earned maturity. These stICP tokens can be used like regular ICP to earn additional yields and lending rewards. The value of stICP increases relative to ICP over time, as the protocol canisters manage the staked ICP and earn NNS voting rewards. The ratio of stICP to ICP is updated approximately every 24 hours based on these rewards, ensuring that the value of stICP reflects the accumulated staking rewards.

The protocol utilizes a collection of neurons within the Network Nervous System (NNS) to stake the deposited ICP. These neurons, managed by the protocol canisters, have various staking delays ranging from 6 months to 8 years, and are rebalanced to maintain liquidity and yield. This setup ensures that users can benefit from staking without being locked into long-term commitments.

Key Features

Auto-Compounding: ICP is staked for up to 8 years, with interest accruing daily to maximize returns. Users automatically receive the benefits of staking by holding stICP tokens. No Lock-in: Users can sell their stICP for ICP at any time, providing liquidity.

Liquid Staking: Users receive stICP tokens that can be used like regular ICP to earn yields and lending rewards. The value of stICP increases relative to ICP based on NNS voting rewards.

Neuron Management: The protocol manages a collection of neurons with varying staking delays, ensuring liquidity and yield optimization.

Future Governance: The protocol aims to transition to full community governance through a DAO model.

1.2 Scope

1.2.1 In Scope

The audit was performed for the Github repository: icp-StakedICP .

Initial Commit Hash : `cb904515c07b590deb9a22538cb198adfeae3a8a`

Within Scope

1. Deposits Canister: The Deposits canister manages several subsystems related to the StakedICP protocol, including referrals, neurons, staking, withdrawals, and background job processing. It interacts with other canisters like governance, ledger, and token to maintain and update state across upgrades, track pending transfers, and handle various operations.

Owner Management:

- Add and remove owners.
- Retrieve the list of all owners.

Staking Management

- Manage staking neurons.
- Reset and update staking neurons.
- Retrieve staking neuron balances.

Deposit and Withdrawal Management

- Provide deposit addresses and subaccounts.
- Process ICP deposits by minting stICP tokens.
- Create and complete withdrawals, ensuring sufficient liquidity

Metrics and Reporting

- Calculate and provide exchange rates and available liquidity

Daily and Heartbeat Tasks

- Perform daily operations like applying interest, flushing pending deposits, and splitting withdrawal neurons
- Execute periodic tasks like refreshing balances and token supply

Upgrade and Recovery

- Manage state during upgrades to ensure data stability.
- Provide functions for manual recovery in case of issues with recent operations

2. Metric Canister

The Metrics canister is designed to gather and serve various metrics related to the StakedICP protocol, providing insights into the protocol's performance and state.

- Collects and aggregates metrics from the Deposits, Token, and Signing canisters.
- Provides detailed information such as token supply, transaction history, number of holders, and canister balances.
- Calculates and reports the Total Value Locked (TVL) in the deposits canister
- Periodically refreshes and updates metrics data to ensure accuracy and timeliness

3. Signing Canister

The Signing canister is designed to manage and perform ECDSA operations like signing messages. It also maintains ownership information and ensures that only authorized users can perform sensitive operations.

- Allows adding and removing owners. Only existing owners can modify the ownership list.
- Signs a message using the specified ECDSA key.

1.2.2 Out of Scope

1. cmd/Oracle: This is an off-chain scripts that runs on a daily basis to call various authorized functions on the deposits canister.

2. Website Canister & Front-end Integration: The security of the integration between the front end and the back end was outside of the scope of this audit.

3. Testing: The auditors review and audit existing tests in the client repository, but it's outside of the scope of this audit for the auditors to develop new tests for the client repository.

4. Internal modules & libraries

- NNS Governance & NNS Ledger related interfaces.
- Binary.mo
- CRC32.mo
- Hex.mo
- SHA224.mo
- SHA256.mo These are system modules and their security or integration was not part of the audit.

1. Protocol-level Audit: Auditing the entire Internet Computer Protocol (ICP), and the related tooling/packages provided by DFINITY Foundation, was outside of the scope of this audit.

2. External Libraries: Motoko Base Libraries and DIP-20 Contract are external libraries and system modules, and they did not receive an independent in-depth audit evaluation outside of their usage context within the scoped canisters.

1.3 Centralization Risks Analysis

Access Controls and Ownership in StakedICP In the StakedICP protocol, access controls play a critical role in maintaining security and ensuring that only authorized entities can perform specific operations. There are two primary categories of functions:

- **Functions Callable by Users**: These functions are designed for end-users and allow them to interact with the protocol, such as staking ICP, withdrawing staked tokens, and checking their balances.
- **Functions Restricted to Owners**: These functions are accessible only to the owners of the canisters and involve critical operations that require a higher level of control and security.

Owners' Controls The owners of the StakedICP canisters have significant control over various aspects of the protocol. Their responsibilities and powers include, but are not limited to:

- **Adding and Removing Owners**: Owners can add new owners or remove existing ones, thereby managing the group of individuals who have high-level control over the protocol.
- **Setting Token and Managing Staking Neurons**: Owners can set the principal of the stICP token and manage staking neurons, including resetting the list of neurons, adding new neurons, and managing neuron states.
- **Flushing Pending Deposits**: Owners can trigger the function to flush pending deposits, ensuring that deposits are processed and routed correctly within the protocol.
- **Setting APR Override**: Owners can override the Annual Percentage Rate (APR) for staking rewards, allowing them to adjust the reward rates as necessary.
- **Handling Neuron Disbursals and Withdrawals**: Owners can list and manage neurons that are ready for disbursal, as well as those being dissolved to fulfill withdrawal requests.
- **Processing ICP Deposits**: Owners can process user deposits, minting the equivalent amount of stICP tokens.
- **Managing Daily Jobs**: Owners can execute daily jobs that include refreshing neurons, applying interest, and ensuring the protocol operates smoothly.
- **Upgrading and Maintenance**: Owners have the authority to upgrade the protocol, set referral data, manage job results, and pause the scheduler for maintenance purposes.

Neuron Management in StakedICP

Overview of Neuron Controls In the StakedICP protocol, neurons play a pivotal role in the staking mechanism. Neurons are entities within the Internet Computer Network Nervous System (NNS) that allow users to participate in governance, earn rewards, and manage their staked ICP tokens. The protocol has a robust system for adding and managing neurons to ensure the security and efficiency of the staking process.

- **Neuron Creation**: The creation of neurons typically involves generating a neuron ID within the NNS system. The owners initiate the creation process by setting the controller and sending ICP to the neuron account.
- **Claiming the Neuron**: Once the neuron is funded, the owners call the NNS governance manage neuron function to claim the neuron.
- **Control and Management**: Although the Deposits canister has control over the neurons via the hot key, the actual controller of the neuron is set during the neuron account ID creation and is typically the owner.
- **Integration with Staking System**: After claiming, the neuron is integrated into the staking management system within the Deposits canister. The neuron details are refreshed and tracked, allowing the protocol to manage rewards and governance participation effectively.

Centralization Risks in StakedICP

Access Control and Management Risks The StakedICP protocol's reliance on owners for crucial operations introduces several centralization risks. The protocol's overall access control management is heavily dependent on the actions and integrity of its owners. Here are the primary concerns:

- **Owner Dominance**: Since the protocol requires owners to perform critical functions, a rogue owner can have a significant impact on the protocol's stability. This includes the ability to add or remove other owners, effectively gaining unilateral control over the protocol.
- **Operation Disruption**: A rogue owner can disrupt essential daily operations, such as flushing deposits or executing daily interest calculations. Failure to perform these actions can lead to user deposits not being added to staking, improper handling of user withdrawals, and overall chaotic protocol behavior.
- **Sensitive Function Manipulation**: Owners have access to sensitive functions that can alter key protocol parameters. For example, setting applied interest or total maturity can directly impact the rewards distribution. Unauthorized changes to these parameters can lead to unintended financial outcomes and erode user trust.
- **Staking Neuron Management**: Owners also manage the addition and removal of staking neurons. If an owner decides to remove all other owners and modify the neuron configurations, it can cause severe disruptions, including loss of staking rewards or improper neuron management.

Neuron Control Risks The management of neurons within the StakedICP protocol also presents centralization risks due to the control exerted by the owners. Here are the key concerns:

- **Controller Influence**: The controller of a neuron is set during its creation, and this controller is typically one of the owners. This setup means that owners can make changes to neuron configurations, potentially impacting the staking rewards and governance participation.
- **Unnecessary Changes**: Rogue owners can make unnecessary or malicious changes to neuron settings. This can include altering the neuron's hot key or other configurations, leading to instability in staking operations and potential financial losses for users.
- **Protocol Integrity**: The ability to control neuron configurations means that owners have significant influence over the protocol's operation. Any malicious action by an owner can compromise the protocol's integrity, leading to loss of user confidence and potential financial harm.

1.4 Methodology

The codebase was audited using a filtered audit technique. A band of two (2) auditors scanned the codebase in an iterative process for a time spanning 3.5 Weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations and security design patterns, code styles and best practices.

Questions for Security Assessment

The engagement is scoped to conduct an exhaustive security assessment of the targeted StakedIcp Canisters. We aim to address the following non-exhaustive list of questions, each targeting fundamental security concerns:

1. Are stable variables used appropriately, and are pre-upgrade and post-upgrade hooks implemented to handle state migration?
2. What mechanisms are in place to handle traps, and how are they prevented from affecting the actor's state?
3. Are arithmetic operations safe from overflow and underflow issues?
4. What safeguards are present to prevent cycle drainage attacks?
5. Can users manipulate the system to gain more rewards, and how is this prevented?
6. What measures are in place to prevent DoS attacks, both intentional and unintentional?
7. Are withdrawal processes robust and handle all edge cases effectively?
8. Are async calls managed correctly, ensuring proper inter-canister communication?
9. Is access control enforced throughout the canister's functionality to prevent unauthorized actions?
10. Is the upgrade process well-designed, with proper handling of state migration and adherence to best practices?
11. Are there any risks of resource exhaustion?

1.5 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.6 Summary of Findings Identified

S.No	Severity	Findings	Status
#1	CRITICAL	Loss of Protocol Funds (Stlcp) Due to Inability to Withdraw Remaining Amounts.	FIXED
#2	CRITICAL	Potential Resource exhaustion & cycle drainage	FIXED
#3	HIGH	Cycle Drainage Vulnerability Due to Missing Inspect Function	FIXED
#4	MEDIUM	Initial ICP Deposit Requirement Leading to Potential Financial Loss	ACKNOWLEDGED
#5	MEDIUM	Initial Deposit Manipulation Leading to Exchange Rate Exploitation	ACKNOWLEDGED
#6	LOW	Risk of Response Size Exceedance Due to Lack of Pagination in Large Dataset Functions	CLOSED
#7	INFO	Double Counting in flushAllMints Function Causes Incorrect Funds Display	CLOSED
#8	INFO	Deprecated Functions Should Be Removed	FIXED
#9	INFO	Unauthorized Access to Neuron Account IDs Due to Ineffective Owner Check	FIXED
#10	INFO	Build Failure Due to Duplicate Function Declarations	FIXED
#11	INFO	Outdated Compiler and Libraries Lead to Potential Security and Compatibility Issues	CLOSED

2 Findings and Risk Analysis

2.1 Loss of Protocol Funds (StIcp) Due to Inability to Withdraw Remaining Amounts.

Severity: Critical

Status: Fixed

Location :

1. `src/deposits/Daily/ApplyInterest.mo`

Description In the StakedICP protocol, when users make deposits through referrals, 2.5% of each reward from a lead is allocated to the affiliate. The remaining amount is then distributed to the root (deposit canister). However, there is no mechanism in place to withdraw the residual amount from the deposit canister, leading to a scenario where these funds are effectively lost. This issue arises because the protocol lacks a function that allows the withdrawal of these accumulated funds from the deposit canister.In the StakedICP protocol, when users make deposits through referrals, 2.5% of each reward from a lead is allocated to the affiliate. The remaining amount is then distributed to the root (deposit canister). However, there is no mechanism in place to withdraw the residual amount from the deposit canister, leading to a scenario where these funds are effectively lost. This issue arises because the protocol lacks a function that allows the withdrawal of these accumulated funds from the deposit canister.

Impact The inability to withdraw remaining amounts from the deposit canister results in a direct financial loss to the protocol. These unclaimed funds could have been used for other purposes within the protocol, such as further development or rewarding participants.

Proof of Concept

```
// Protocol takes the remainder. For now, just mint it to this
           // canister.
           if (remainder &gt; 0) {
               mints.add((root, remainder));
               remainder := 0;
           };
```

Recommendation Introduce a function within the Deposits canister that allows the withdrawal of the remaining StICP. This function should ensure that all accumulated amounts can be accessed and utilized appropriately.

The issue has been fixed in this PR#2

2.2 Potential Resource exhaustion & cycle drainage

Severity: Critical

Status: Fixed

Location :

1. `src/deposits/deposits.mo`

Description There exist functions like `createWithdrawal` and `getReferralStats` that open a window of opportunity for attackers to inflate the system storage and cause upgrade issues due to numerous entries. These functions can potentially cause a cycle drainage attack.

1. **Small Withdrawals Leading to Resource Exhaustion:** Users can exploit the system by making many very small withdrawals. Each withdrawal requires the canister to perform several operations, such as checking balances, burning tokens, and updating states. When users flood the system with these tiny transactions, it can significantly inflate state storage and place an excessive load on the canister's computation cycles.
2. **Excessive Referral Code Generation:** The `getReferralStats` function allows users to generate referral codes for their accounts. This function generates a new code the first time it is called for an affiliate and stores it. Users can exploit this by creating numerous referral codes without any cost, leading to resource exhaustion. The collision handling mechanism, which involves regenerating codes until a unique one is found, can be especially taxing.

Impact :

Denial of Service (DoS): Both small withdrawals and excessive referral code generation can lead to excessive resource consumption. The heavy load from countless small transactions or numerous referral code generations can quickly deplete the canister's computation cycles.

Upgrade Challenges: During upgrades, the canister needs to migrate its state, including all stored withdrawal transactions and referral codes. A large number of small, pending withdrawals or numerous generated referral codes can complicate this process, increasing the risk of errors and incomplete migrations.

Recommendation :

1. **Rate Limiting**: Limit the number of referral code generations and withdrawal requests per user over a specific time period to prevent abuse.
2. **Minimum Withdrawal Amount**:Introduce a minimum amount for withdrawals to prevent users from creating many small transactions. This can reduce the risk of resource exhaustion.
3. **Introduce a Cost for Generating Codes**:Charge a small fee for generating referral codes to discourage users from creating an excessive number of codes.

The issue has been fixed in this PR#3

2.3 Cycle Drainage Vulnerability Due to Missing Inspect Function

Severity: High

Status: Fixed

Location :

The issue exists in the overall architecture of the StakedICP protocol

Description The StakedICP protocol is vulnerable to cycle drainage attacks due to the absence of the `inspect` function. This function plays a crucial role in filtering and validating incoming messages before they are processed by the canister. Without it, the canister is exposed to potential attacks that can exhaust its computation cycles, leading to a Denial of Service (DoS).

The canister does not currently implement the `inspect` method, which could be used to authenticate messages quickly and efficiently before they consume significant computational resources. This vulnerability opens the door for attackers to send numerous unauthenticated or malicious requests, draining the cycle balance and potentially rendering the canister inoperative

Impact :

1. **Denial of Service (DoS)**: Attackers can deplete the canister's cycles by sending a high volume of requests. This would result in the canister running out of cycles, making it unable to process legitimate transactions.
2. **Resource Exhaustion**: The lack of preemptive checks allows malicious actors to inflate the system's state storage and computation requirements, further exacerbating the cycle drainage issue.
3. **Operational Downtime**: If the canister runs out of cycles, it will stop functioning until manually recharged, leading to potential service interruptions and user dissatisfaction.

Recommendation :

1. **Implement the Inspect Function**: Integrate the inspect function to perform early validation of incoming messages. This function should quickly authenticate requests and reject those that do not meet the necessary criteria, thereby conserving cycles
2. **Cycle Monitoring**: Regularly monitor the cycle balance of the canister to ensure it remains above the freezing threshold. Automated alerts can be set up to notify administrators when the cycle balance falls below a certain level.
3. **Rate Limiting and Throttling**: Implement rate limiting for functions that can be called frequently to prevent abuse. This involves setting limits on the number of times a function can be called within a specific time frame.

The issue has been fixed in this PR#4

2.4 Initial ICP Deposit Requirement Leading to Potential Financial Loss

Severity: Medium

Status: Acknowledged

Location :

1. `src/deposits/deposits.mo`

Description The deposit process in the StakedICP protocol necessitates an initial deposit of ICP to enable the first user deposits. This initial deposit is required to ensure that the total ICP in the canister is greater than zero. However, this requirement leads to several issues:

1. **Loss of Initial ICP**: The initial ICP deposited to the canister's subaccount to enable the first deposits will be effectively lost, as there is no mechanism to reclaim this amount. This results in a permanent loss of the deposited ICP, which can be financially significant.
2. **Exchange Rate Discrepancies**: The need to maintain a 1:1 exchange rate between ICP and stICP tokens means that an equivalent amount of stICP must be minted corresponding to the initial ICP deposit. This creates a situation where the total ICP in the system is artificially inflated, leading to potential inaccuracies in the exchange rate calculation and discrepancies in the financial model.

Recommendation Introduce a dedicated mechanism for handling the initial ICP deposit. This could involve a temporary funding mechanism or a system to reclaim the initial deposit once user deposits are enabled.

2.5 Initial Deposit Manipulation Leading to Exchange Rate Exploitation

Severity: Medium

Status: Acknowledged

Location :

1. `src/deposits/deposits.mo`

Description In the StakedICP deposits canister, there exists a vulnerability where the first depositor can manipulate the exchange rate between stICP and ICP tokens. This manipulation can significantly impact subsequent deposits and the overall tokenomics of the protocol.

The exchangeRate function calculates the exchange rate based on the total supply of stICP and the available ICP assets. When the first deposit is made, it sets the baseline for this exchange rate. An attacker can exploit this by making an initial deposit of a small amount of ICP, thereby controlling the initial exchange rate. Subsequent deposits would then be subject to this manipulated rate, allowing the attacker to inflate the supply of stICP tokens disproportionately.

Impact :

1. **Inflation Attack**: An attacker can set an artificially low initial exchange rate, causing stICP to be minted excessively for subsequent deposits, inflating the token supply.
2. **Economic Imbalance**: Early depositors can benefit disproportionately, receiving more stICP than the fair value, leading to significant economic disparities within the protocol.

Recommendation :

1. **Initial Sacrifice Mechanism**: Introduce a mechanism where the initial liquidity provider must sacrifice a portion of their stICP tokens (e.g., 1000 tokens) to address potential manipulation and ensure granularity in the exchange rate.

2. **Weighted Average Approach**: Use a weighted average calculation for the exchange rate based on multiple initial deposits, rather than a single deposit, to stabilize the rate.

2.6 Risk of Response Size Exceedance Due to Lack of Pagination in Large Dataset Functions

Severity: Low

Status: Closed

Location :

1. `src/deposits/deposits.mo`

Description The StakedICP protocol has several functions that return large datasets. These functions include `listWithdrawals`, `getReferraldata`, and `metrics`. As the data grows, the response size from these functions can exceed the maximum allowable response size, leading to incomplete data retrieval and potential function execution failures. This issue arises because the current implementation does not paginate the returned data, resulting in the entire dataset being sent in a single response.

Impact :

1. **Incomplete Data Retrieval**: When the dataset grows large, the response might exceed the maximum response size limit, causing only a portion of the data to be retrieved.
2. **User Experience**: Users may experience delays or errors when attempting to retrieve large datasets

Recommendation Implement pagination for functions that return large datasets to ensure consistent performance and data integrity

2.7 Double Counting in flushAllMints Function Causes Incorrect Funds Display

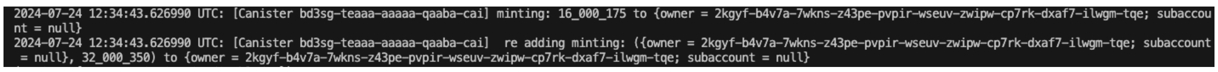
Severity: Info

Status: Closed

Location :

1. `src/deposits/deposits.mo`

Description The `flushAllMints` function is designed to execute all pending mints on the token canister. However, there is a flaw in the error handling logic. When the mint operation fails, the pending mints are re-added to the pendingMints TrieMap without clearing the previously failed entries. This leads to a situation where the rewards are counted twice in the system, even though they are not actually minted twice. This discrepancy can cause confusion and incorrect reward displays to users.



```
2024-07-24 12:34:43.626990 UTC: [Canister bd3sg-teaaa-aaaaa-qaaba-cai] minting: 16_000_175 to {owner = 2kgyf-b4v7a-7wkns-z43pe-pvpir-wseuv-zwipw-cp7rk-dxaf7-ilwgm-tqe; subaccou
nt = null}
2024-07-24 12:34:43.626990 UTC: [Canister bd3sg-teaaa-aaaaa-qaaba-cai]  re adding minting: ({owner = 2kgyf-b4v7a-7wkns-z43pe-pvpir-wseuv-zwipw-cp7rk-dxaf7-ilwgm-tqe; subaccount
 = null}, 32_000_350) to {owner = 2kgyf-b4v7a-7wkns-z43pe-pvpir-wseuv-zwipw-cp7rk-dxaf7-ilwgm-tqe; subaccount = null}
```

Recommendation To resolve this issue, the error handling logic in the `flushAllMints` function should be corrected to ensure that failed minting operations do not result in double counting of funds.

2.8 Deprecated Functions Should Be Removed

Severity: Info

Status: Fixed

Location :

1. `src/deposits/deposits.mo`

Description The Deposits canister code includes two functions, `proposalNeuron` and `setProposalNeuron`, which are deprecated and no longer in use. The proposalNeuron function always returns null, and the setProposalNeuron function immediately returns an error indicating that the proposal neuron feature is deprecated. These functions serve no purpose in the current implementation

Affected Code

```
public shared(msg) func proposalNeuron(): async ?Neurons.Neuron {
        null
    };

    public shared(msg) func setProposalNeuron(id: Nat64): async Neurons.NeuronResult {
        owners.require(msg.caller);
        #err(#Other(&quot;Proposal neuron is deprecated.&quot;))
    };
```

Recommendation The deprecated `proposalNeuron` and `setProposalNeuron` functions should be removed from the codebase

The issue has been fixed in this PR#5

2.9 Unauthorized Access to Neuron Account IDs Due to Ineffective Owner Check

Severity: Info

Status: Fixed

Location :

1. `src/deposits/deposits.mo`

Description The `neuronAccountId` function in the Deposits canister code is intended to be an owner-only function that returns the neuron account ID for a given controller and nonce. The function enforces an ownership check using `owners.require(msg.caller);`. However, the required information to compute the neuron account ID is publicly accessible. Any user can obtain the governance principal and controller information through the `get_all_owners`function and other known arguments, rendering the owner check redundant and ineffective

Affected Code:

```
public shared(msg) func neuronAccountId(controller: Principal, nonce: Nat64): async Text {
    owners.require(msg.caller);
    return NNS.accountIdToText(Util.neuronAccountId(args.governance, controller, nonce));
};
```

Impact Any user can determine neuron account IDs without being an owner, allowing actions such as sending funds to or claiming neurons on behlaf of the controller.

Recommendation Remove the ownership check from the `neuronAccountId` function to avoid the false security implication.

The issue has been fixed in this PR#6

2.10 Build Failure Due to Duplicate Function Declarations

Severity: Info

Status: Fixed

Location :

1. `src/signing/src/lib.rs`

Description The codebase contains duplicate declarations for the `add_owner` and `remove_owner` functions, which leads to a build failure. The error arises from defining these functions multiple times within the same module. The compiler explicitly flags this issue, preventing the code from compiling successfully.

Recommendation Remove the redundant declarations of the `add_owner` and `remove_owner`functions. Ensure that each function is defined only once within the module.

The issue has been fixed in this PR#7

2.11 Outdated Compiler and Libraries Lead to Potential Security and Compatibility Issues

Severity: Info

Status: Closed

Description The current protocol uses outdated versions of the Motoko compiler and various libraries, which can lead to several issues, including security vulnerabilities, compatibility problems, and maintenance challenges. The Motoko compiler version being used is `0.6.21`, while the latest version available is `0.11.1`. Additionally, several npm packages are outdated, exposing the project to known vulnerabilities and bugs.

Recommendation Update the Motoko Compiler and upgrade NPM packages

2.12 Decentralization Readiness Findings

2.12.1 Transition to Single Owner for SNS Governance

Severity: Currently Medium, High for SNS

Status: Closed

Description: The `stakedICP` protocol's canister currently includes an authorization list of principals that can perform significant administrative actions. For true decentralization and readiness for SNS (Service Nervous System) governance, the canister should transition to a single owner model. This single owner should eventually be the SNS governance canister to ensure that all critical interactions are managed through SNS governance proposals. The methods impacted include `addOwner`, `removeOwner`, `setToken`, `setAprOverride`, `setMetrics`, and similar administrative functions.

Impact: Maintaining a list of authorized principals post-SNS launch could compromise decentralization. Even if the canister's ownership is transferred to the SNS governance canister, any principal in the authorization list could still execute administrative actions, leading to potential manipulation and centralized control risks.

Recommendation: Remove the list of authorized principals and retain only the owner field. This owner field should be updated to point to the SNS governance canister once the SNS is launched. Ensure that the ownership is verifiable through the canister's API to maintain transparency and trust.

2.12.2 Expose Query Methods to Monitor Canister Status

Severity: Currently Medium, High for SNS

Status: Closed

Description: The `stakedICP` protocol's canisters do not currently provide query methods to expose internal status and metrics such as available cycles, canister size, used stable memory and other relevant information. Once these canisters are managed by the SNS governance canister, it will be critical to have such methods in place to allow for effective monitoring and transparency.

Impact: Without these query methods, it will become challenging to monitor the canisters' status and performance after ownership is transferred to the SNS governance canister. This lack of visibility could hinder effective governance and operational oversight.

Recommendation: Implement query methods in all relevant canisters to expose internal information, such as: - Available cycles - Canister size - Used stable memory - Other relevant metrics

This will ensure that even under SNS governance, the necessary information for monitoring and transparency is readily available.

Disclaimer:

The smart contracts (canisters) provided by the client for security review have been thoroughly analyzed in compliance with the best industrial practices to date concerning cybersecurity vulnerabilities in smart contract code. The details of this analysis are enclosed in this report.

This report is not an endorsement or indictment of the project or team and does not guarantee the security of the particular object in context. It should not be interpreted as influencing the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing. Crypto assets/tokens are the results of the emerging blockchain technology in the domain of decentralized finance and carry high levels of technical risk and uncertainty.

No report provides any warranty or representation to any third party in any respect, including regarding the bug-free nature of code, the business model, or the legal compliance of any such business. No third party should rely on the reports in any way, including making decisions to buy or sell any token, product, service, or asset. Specifically, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee of the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code only, as noted within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review 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 BlockApex has done its best in conducting the analysis and producing this report, it is important to note that one should not rely on this report alone. We recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.

ICP decentralized app security is a constantly changing new frontier full of unknowns. Any potential users of DeFi dapps on ICP should keep this in mind and do their own research. Neither BlockApex nor our auditors are responsible for any losses experienced by users of the client protocol. There is no such thing as a comprehensive audit, but there is always a limited scope. The work performed for this audit is captured within this report. Readers of this report should not assume that anything not explicitly covered here was evaluated.

Due to the experimental nature of audits for ICP applications, neither BlockApex nor the auditors make any guarantees regarding the security of the client protocol. We conducted a best-effort audit in good faith with talented protocol experts, but that is not enough to guarantee that new exploits won't ever be discovered on a technology as novel as ICP. ICP canisters have mutable code. This audit report was conducted on a specific commit hash of the client protocol, and the issues were verified to be fixed on another specific hash. Once the code is updated again, the applicability of this audit is voided and incremental reviews are recommended, as future versions of the client protocol source code may introduce new vulnerabilities.