SECURITY ASSESSMENT

Accretion

Provided by Accretion Labs Pte Ltd. for Sanctum
November 06, 2025
A25SAN1

Sanctum

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	David Ratiney (david@accretion.xyz)

CLIENT

Sanctum (https://sanctum.so) is a project building Solana staking infrastructure such as liquid staking tokens, swaps between such tokens, instant unstake, and more. They engaged Accretion to conduct a security assessment of three parts: The Jiminy solana SDK, the token-ratio library and a PR to their pricing program.

ENGAGEMENT TIMELINE

08 Sep — Project Kickoff
Initial planning and scope definition

08 Sep — Assessment Begins
Security review and testing phase

15 Oct — Review Fixes
Security recommendations are given and implemented

25 Oct — Project Completion
Report delivery and on-chain confirmation

AUDITED CODE

Program 1
ProgramID: N/A
Repository: https://github.com/igneous-labs/jiminy

Program 2
ProgramID: N/A
Repository: https://github.com/igneous-labs/sanctum-token-ratio

Program 3
ProgramID: s1b6NRXj6ygNu1QMKXh2H9LUR2aPApAAm1UQ2DjdhNV
Repository: https://github.com/igneous-labs/inf-1.5/pull/50

ASSESSMENT

The security assessment covered three components of Sanctum's products: two libraries and one pull request to an existing program. The review revealed exceptionally well-built implementations across all components, yielding only six findings ranging from low to medium severity with no high or critical issues identified.

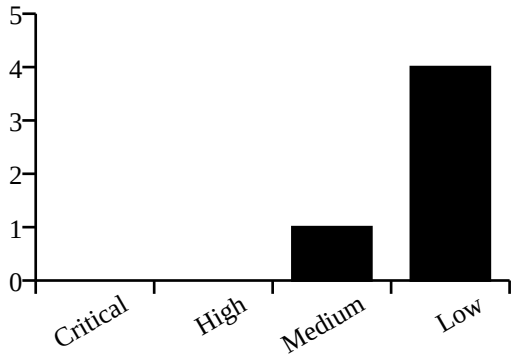
CODE ASSESSMENT

The code quality was very good across all programs reviewed. The codebase demonstrates a unique style that heavily relies on advanced Rust features and coding patterns, frequently employing generics and functional programming paradigms. A thorough test suite was provided with the code, though documentation quality, while adequate, was on the lower end and could benefit from additional detail.

KEY FINDINGS

Six issues were identified during the audit, with only one classified as medium severity. The medium severity issue was raised due to the Jiminy framework missing validation checks for calls to the system program. The remaining five low or info severity findings consisted primarily of minor validation checks and offset corrections. All issues have been fixed.

SEVERITY DISTRIBUTION



© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: Jiminy Framework
Link: https://github.com/igneous-labs/jiminy
Commit: a4f1381ed9291404d32c889be019a1831d0583cc
Program ID: N/A
Audit Result:
- **Audited Commit:** 7376cce206f7d45c64cdc641fb621811ada65731
- **Build Hash:** N/A
- **Status:** N/A
- **Comment:** Verification not applicable (library)

Item 2: Token Ratio
Link: https://github.com/igneous-labs/sanctum-token-ratio
Commit: d6c859be4c1420a781fc12f70f6f2a721da6ba1e
Program ID: N/A
Audit Result:
- **Audited Commit:** 08c8ccc1847b41939e2170526ee9541c34082ccc
- **Build Hash:** N/A
- **Status:** N/A
- **Comment:** Verification not applicable (library)

Item 3: inf-1.5 PR 50
Link: https://github.com/igneous-labs/inf-1.5/pull/50
Commit: e0e34574726e60b50cebb827c3397e5082952f8d
Program ID: s1b6NRXj6ygNu1QMKXh2H9LUR2aPApAAm1UQ2DjdhNV
Audit Result:
- **Audited Commit:** d22d6e90733bc95ca4bc7f51b0cfb90dfbc59e36
- **Build Hash:** 47ad0865fa5ac15a3ec46412312859e6517494540c40fe8ad9f3b2f72e1827d1
- **Status:** Verified
- **Comment:** Program hash verified on-chain

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-M1	No system program address validation	medium	fixed
ACC-L1	The deserializer reads uninitialized memory	low	fixed
ACC-L2	`LP_MINT_ID` can be removed from the slab	low	fixed
ACC-L3	Lack of constraint in `set_lst_fee`	low	fixed
ACC-L4	`fee-ratio` - Zero-denominator treated as zero ratio permits silent fee bypass	low	fixed
ACC-I1	Is full can return false on over-filled seed array	info	fixed

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

DETAILED ISSUES

ID	ACC-M1
Title	No system program address validation
Severity	medium
Status	fixed

Description

We found that the jiminy framework does not automatically verify the system program to be invoked when construction instructions for the system program. If a user passes a program that is not the system program, this may lead to critical issues. This is evident in the example `cpi-sys-transfer` program, where a fake system program will still result in a CPI.

Location

https://github.com/igneous-labs/jiminy/blob/7b3b884eb7144d626e4f6e4d3881d9ea44bab150/test-programs/cpi-sys-transfer/src/lib.rs#L25-L42

Relevant Code

```
/// lib.rs L25-L42
    let (sys_prog, transfer_accs) = match accounts.as_slice().split_last_chunk() {
        Some((&[sys_prog], ta)) => (sys_prog, TransferIxAccs(*ta)),
        _ => {
            return Err(ProgramError::from_builtin(
                BuiltInProgramError::NotEnoughAccountKeys,
            ))
        }
    };

    let [from_lamports_bef, to_lamports_bef] =
        [transfer_accs.from(), transfer_accs.to()].map(|handle| accounts.get(*handle).lamports());

    Cpi::<MAX_CPI_ACCS>::new().invoke_fwd_handle(
        accounts,
        sys_prog,
        TransferIxData::new(trf_amt).as_buf(),
        transfer_accs.0,
    )?;
```

Mitigation Suggestion

There should be an API that ensures the correct system program is used. For example when `TransferIxAccs` is invoked like this:

```
    let (sys_prog, transfer_accs) = match accounts.as_slice().split_last_chunk() {
        Some((&[sys_prog], ta)) => (sys_prog, TransferIxAccs(*ta)),
```

There should be an alternative way which also validates the sys_prog here.

Remediation

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

Fixed in commit `66c0a4a7539892d588ccf08f8ecee1c82b9f09ed`.

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ID	ACC-L1
Title	The deserializer reads uninitialized memory
Severity	low
Status	fixed

Description

We found that the deserializer reads uninitialized memory when a duplicate account in the truncated tail (beyond `MAX_ACCOUNTS`) is processed (for instance `MAX_ACCOUNTS = 8` and the following account slice is passed `[A0..A7, X, X]`)

Location

https://github.com/igneous-labs/jiminy/blob/a4f1381ed9291404d32c889be019a1831d0583cc/account/src/deser.rs#L75

Relevant Code

```
let input = (saved_accounts_len..accounts_len).fold(input, |input, _| match input.read() {
        NON_DUP_MARKER => AccountHandle::non_dup_from_ptr(input, &accounts).0,
        dup_idx => AccountHandle::dup_from_ptr(input, dup_idx, &accounts).0,
    });
```

Mitigation Suggestion

```
  let input = (saved_accounts_len..accounts_len).fold(input, |input, _| match input.read() {
        NON_DUP_MARKER => AccountHandle::non_dup_from_ptr(input, &accounts).0,
        _ => unsafe { input.add(8) }
    });
```

Remediation

Fixed in `730b52db84813a2be260822ded7a882db71426ac`

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ID	ACC-L2
Title	`LP_MINT_ID` can be removed from the slab
Severity	low
Status	fixed

Description

We found that the `remove_lst` instruction does not prevent the admin from removing by mistake the `LP_MINT_ID`. If this occurs, the protocol's LP mint would be deleted from the slab, breaking the functionalities of the controller program that depends on it.

Location

https://github.com/igneous-labs/inf-1.5/blob/42497f7fba2cfd03d9e5d05f58b2be609b6b53cc/pricing/flatslab/program/src/instructions/admin/remove_lst.rs#L47

Relevant Code

Mitigation Suggestion

To prevent this misconfiguration, the instruction should explicitly reject attempts to remove the `LP_MINT_ID`.

Remediation

Fixed in `467842471a15a6326a9a1d2aef86c58e9eec8f7d`

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ID	ACC-L3
Title	Lack of constraint in `set_lst_fee`
Severity	low
Status	fixed

Description

We found that there is no validation on the values of `inp_fee_nanos` and `out_fee_nanos`. This leaves room for admin misconfiguration (for example, entering values in bps instead of nanos) or, in the case of admin key compromise, malicious configuration that can grief the protocol by breaking quoting behavior.

At a minimum, the program should enforce a constraint to guarantee safe arithmetic operations:

```
1_000_000_000 - i32::MAX ≤ inp_fee + out_fee <= 1_000_000_000
```

For stronger protection against both accidental mistakes and abusive settings something like this (–5 bps to +10%):

```
-50_000 ≤ inp_fee + out_fee <= 100_000_000
```

Location

https://github.com/igneous-labs/inf-1.5/blob/42497f7fba2cfd03d9e5d05f58b2be609b6b53cc/pricing/flatslab/program/src/instructions/admin/set_lst_fee.rs#L49 https://github.com/igneous-labs/inf-1.5/blob/42497f7fba2cfd03d9e5d05f58b2be609b6b53cc/pricing/flatslab/core/src/pricing.rs#L38

Relevant Code

Mitigation Suggestion

Add a constraint on `inp_fee + out_fee` in the `set_lst_fee`

Remediation

Fixed in `824c1ee8eaa4427cc43f1f05b5218e618de7a84e`

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ID	ACC-L4
Title	`fee-ratio` - Zero-denominator treated as zero ratio permits silent fee bypass
Severity	low
Status	fixed

Description

Since `Ratio::is_zero` treats `d == 0` as zero (explicitly documented), `Fee::new(...)` does not reject ratios with a zero denominator. Fees with a zero denominator are effectively 0.

```
    #[test]
    fn fee_with_0_denom() {
        let fee =
            Fee::<ratio::Ceil<ratio::Ratio<u64, u64>>>::new(ratio::Ratio { n: 10, d: 0 }).unwrap();
        let amount = 1_000_000;
        let aft = fee.apply(amount);
        eprint!("{:?}", aft);
    }
```

```
---- tests::x stdout ----
Some(AftFee { rem: 1000000, fee: 0 })
```

Location

https://github.com/igneous-labs/sanctum-token-ratio/blob/d6c859be4c1420a781fc12f70f6f2a721da6ba1e/fee-ratio/src/lib.rs#L72

https://github.com/igneous-labs/sanctum-token-ratio/blob/d6c859be4c1420a781fc12f70f6f2a721da6ba1e/u64-ratio/src/div/ceil.rs#L40

Relevant Code

```
#[inline]
            pub const fn new(fee_ratio: Ratio<$N, $D>) -> Option<Self> {
                if !fee_ratio.is_zero()
                    && fee_ratio.n as <Ratio<$N, $D> as ArithTypes>::Max
                        > fee_ratio.d as <Ratio<$N, $D> as ArithTypes>::Max
                {
                    None
                } else {
                    Some(Self(Ceil(fee_ratio)))
                }
            }


        impl Ceil<Ratio<$N, $D>> {
            /// # Returns
            ///
            /// `ceil(amt * self.0.n / self.0.d)`
            ///
            /// ## Special Case Returns
            /// - `0` if `self.0.is_zero()`
            /// - `None` if `result > u64::MAX`
            #[inline]
            pub const fn apply(&self, amount: u64) -> Option<u64> {
                if self.0.is_zero() {
                    return Some(0);
                }
```

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

```
        let Ratio { n, d } = self.0;
        let d = d as u128;
        let n = n as u128;
        let x = amount as u128;
        // unchecked-arith: mul will not overflow because
        // both x and n are <= u64::MAX
        let xn = x * n;
        // unchecked-arith: ratio is not 0 so d != 0
        let res = xn.div_ceil(d);
        u128_to_u64_checked(res)
    }
```

Mitigation Suggestion

`Fee::new` should reject when `d == 0` regardless of `is_zero`

Remediation

Fixed in `c954acae68d2f0799ba941dd10d03bbb0cc95443`

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

ID	ACC-I1
Title	Is full can return false on over-filled seed array
Severity	info
Status	fixed

Description

We found that the PdaSeedArr implementation has a `push_unchecked` function that allows to push an item to an array, potentially overflowing and thus marked as unsafe. This function increases the length in any case. The `is_full` function checks if the length is exactly `MAX_SEEDS`. When a full array is pushed into, the now overfull array will have a length larger that MAX_SEEDS and the is_full function will return false.

Location

https://github.com/igneous-labs/jiminy/blob/7b3b884eb7144d626e4f6e4d3881d9ea44bab150/pda/src/seed_arr.rs#L49-L68

Relevant Code

```
/// seed_arr.rs L49-L68
    #[inline(always)]
    pub unsafe fn push_unchecked(&mut self, seed: PdaSeed<'seed>) {
        self.seeds[self.len()].write(seed);
        self.len += 1;
    }

    #[inline(always)]
    pub const fn len(&self) -> usize {
        self.len
    }

    #[inline(always)]
    pub const fn is_empty(&self) -> bool {
        self.len() == 0
    }

    #[inline(always)]
    pub const fn is_full(&self) -> bool {
        self.len() == MAX_SEEDS
    }
```

Mitigation Suggestion

We'd suggest changing the implementation to

```
    #[inline(always)]
    pub const fn is_full(&self) -> bool {
        self.len() >= MAX_SEEDS
    }
```

Remediation

Fixed in commit `553f38955c5def7e20f51785b5a3691b122813bc`.

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.

© 2025 Accretion Labs Pte. Ltd. All Rights Reserved.