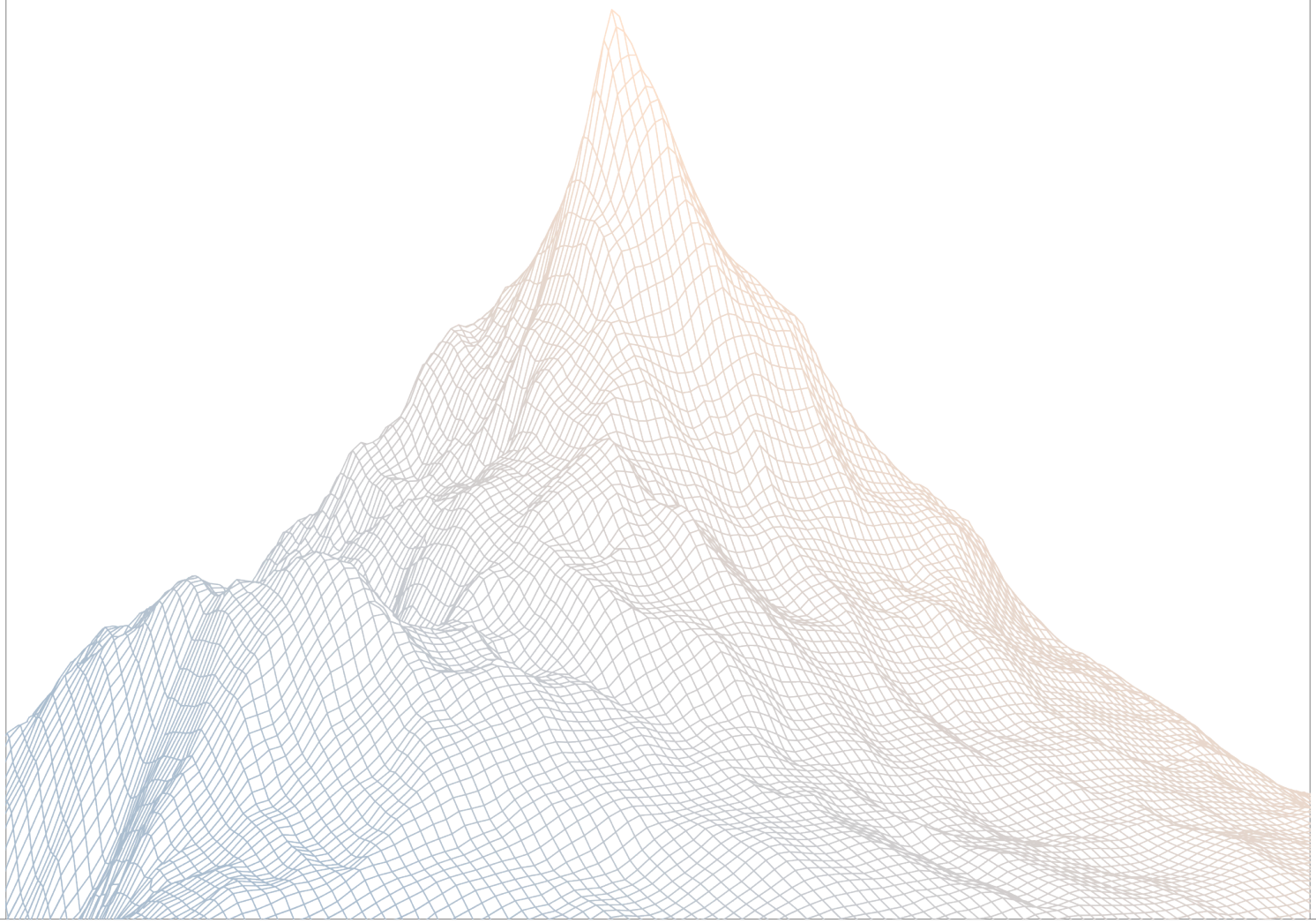
Zenith

SatLayer

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: February 24th to February 5th, 2025

AUDITED BY: Matte
Peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About SatLayer

SatLayer is a shared security platform designed to leverage Bitcoin as the primary security collateral.

By deploying as a set of smart contracts on Babylon, SatLayer enables Bitcoin restakers to secure any type of decentralized application or protocol as a Bitcoin Validated Service (BVS). This means that Bitcoin, the largest and most widely trusted cryptocurrency, can now provide security to a broad range of applications, with full Turing-complete programmability while minimizing trust assumptions.

By expanding the ways Bitcoin can be used, SatLayer helps Bitcoin maximizes its potential as a reward-generating asset.

2.2 Scope

The engagement involved a review of the following targets:

Target	satlayer
Repository	https://github.com/ibriz/satlayer
Commit Hash	`b26e4bedcbc73190ef7716e13b481d91d7e926c7`
Files	`core/sources/*`

2.3 Audit Timeline

February 24, 2025	Audit start
February 25, 2025	Audit end
March 05, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	1
Total Issues	**3**

3

Findings Summary

ID	DESCRIPTION	STATUS
L-1	‘queue_withdrawal()‘ can be mis-used to shorten withdrawal time	Resolved
L-2	‘deposit_for()‘ can be griefed by dust deposits	Resolved
I-1	Freezing of coin metadata for coin ‘LBTC‘	Resolved

4

Findings

4.1 Low Risk

A total of 2 low risk findings were identified.

[L-1] `queue_withdrawal()` can be mis-used to shorten withdrawal time

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- satlayer_pool.move

Description:

`queue_withdrawal()` allows the depositors to request for withdrawal that will be available after `withdrawal_time`. It will set the withdrawal timestamp for the request based on the current withdrawal time. And does the same for existing requests, resetting it as if it is a new withdrawal request.

However, if the withdrawal time is reduced via `update_withdrawal_time()`, users who had a pending withdraw request could actually take advantage of the reset to shorten their withdrawal time. This will be undesirable if its not intended by design.

Example Scenario,

1. At `day 1`, suppose `withdrawal_time = 7 days`.
2. User A calls `queue_withdrawal()`, which allows him to withdraw after `day 7`.
3. At `day 2`, Admin calls `update_withdrawal_time()` to `2 days`.
4. Now, User A sees the update and proceed to deposit and call `queue_withdraw()` a dust amount of coin `T`.
5. User A's existing withdrawal request is reset based on `2 days`. So he can then withdraw the previous requested amount after day 4 instead of day 7.

```
public fun queue_withdrawal<T, K>(vault: &mut Vault<T,K>, receipt_token:
    Coin<K>, clock: &Clock, version:&Version, ctx:&mut TxContext){
```

```
    version.validate_version(VERSION);

    assert!(!vault.is_paused, EVaultIsPaused);
    assert!(receipt_token.value() > 0, EWithdrawalAmountCannotBeZero);

    if(!vault.withdrawal_requests.contains(ctx.sender())) {
        vault.withdrawal_requests.add<address, u64>(ctx.sender(),
    clock.timestamp_ms() + vault.withdrawal_time );
        vault.withdraw_amount.add<address, u64>(ctx.sender(),
    receipt_token.value());
    } else {
        *vault.withdrawal_requests.borrow_mut(ctx.sender()) =
    clock.timestamp_ms() + vault.withdrawal_time;
        let withdraw_amount =
    vault.withdraw_amount.borrow_mut(ctx.sender());
        *withdraw_amount = *withdraw_amount + receipt_token.value();
    };

    event::emit(WithdrawalRequest<K>{
        amount: receipt_token.value(),
        receipt_token_burned: receipt_token.value(),
        withdrawal_timestamp: clock.timestamp_ms(),
    });

    vault.treasury_cap.burn<K>(receipt_token);
}
```

Recommendations:

If required, consider updating the withdrawal timestamp only if the new withdrawal timestamp is later than the existing request's withdrawal timestamp.

SatLayer: Fixed in @bf3af7ff598.

Zenith: Verified. Resolved by only allowing `withdrawal_cooldown` window to be increased and not decreased.

[L-2] `deposit_for()` can be griefed by dust deposits

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- satlayer_pool.move

Description:

Vaults in `SatLayer_pool` can be configured with a `staking_cap` to limit the maximum amount of coin `T` that can be deposited into the vault.

However, the `staking_cap` can be abused by an attacker to grief the last depositor by frontrunning with dust deposits.

Example Scenario,

1. Suppose `vault.balance.value() == 900e9`, and `staking_cap == 1000e9`.
2. Victim calls `deposit_for()` to deposit `100e9` coin `T` into vault.
3. Attacker frontrun victim's tx and deposit `1` (dust amount) into vault.
4. Victim's tx will fail as the `deposit_for()` will now exceed the staking cap by `1`.
5. Attacker can do this repeatedly to grief the victim.

```
public fun deposit_for<T, K>(vault: &mut Vault<T,K>, deposit_amount:
    Coin<T>, version: &Version, ctx: &mut TxContext): Coin<K> {
    version.validate_version(VERSION);

    assert!(!vault.is_paused, EVaultIsPaused);
    assert!(deposit_amount.value() > 0, EDepositAmountCannotBeZero);
    if(vault.caps_enabled && vault.balance.value() + deposit_amount.value()
    > vault.staking_cap) abort ECapReached;

    let deposit_value = deposit_amount.value();

    let balance_before = vault.balance.value();
    vault.balance.join<T>(deposit_amount.into_balance());
```

```
    let actual_amount = vault.balance.value() - balance_before;

    let receipt_coin = vault.treasury_cap.mint<K>(actual_amount, ctx);

    let coin_type = type_name::get<K>();
    event::emit(
       DepositEvent<K>{
        coin_type,
        deposit_amount: deposit_value,
        receipt_token_minted: receipt_coin.value(),
    });

    receipt_coin
}
```

Recommendations:

Consider setting a minimum deposit amount when there is a staking cap, to mitigate the risk of dust deposit attacks.

SatLayer: Fixed in @bf3af7ff598.

Zenith: Verified. Resolved by adding `min_deposit_amount` check when `caps_enabled == true`.

4.2 Informational

A total of 1 informational findings were identified.

[I-1] Freezing of coin metadata for coin LBTC

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- lbtc.move

Description:

It is recommended to freeze the metadata for LBTC, which is a common practice for coins on Sui, as the metadata should almost never change.

```
fun init(witness: LBTC, ctx: &mut TxContext) {
    let (treasury, meta) = coin::create_currency(
        witness,
        9,
        b"lbtc",
        b"",
        b"",
        option::none(),
        ctx,
    );
    transfer::public_share_object(meta);
    transfer::public_transfer(treasury, tx_context::sender(ctx));
}
```

Recommendations:

```
fun init(witness: LBTC, ctx: &mut TxContext) {
    let (treasury, meta) = coin::create_currency(
        witness,
```

```
        9,
        b"lbtc",
        b"",
        b"",
        option::none(),
        ctx,
    );
    transfer::public_share_object(meta);
    transfer::public_freeze_object(meta);
    transfer::public_transfer(treasury, tx_context::sender(ctx));
}
```

SatLayer: Removed file in @7e9fbfbb3e8....

Zenith: Resolved by removing the entire file.