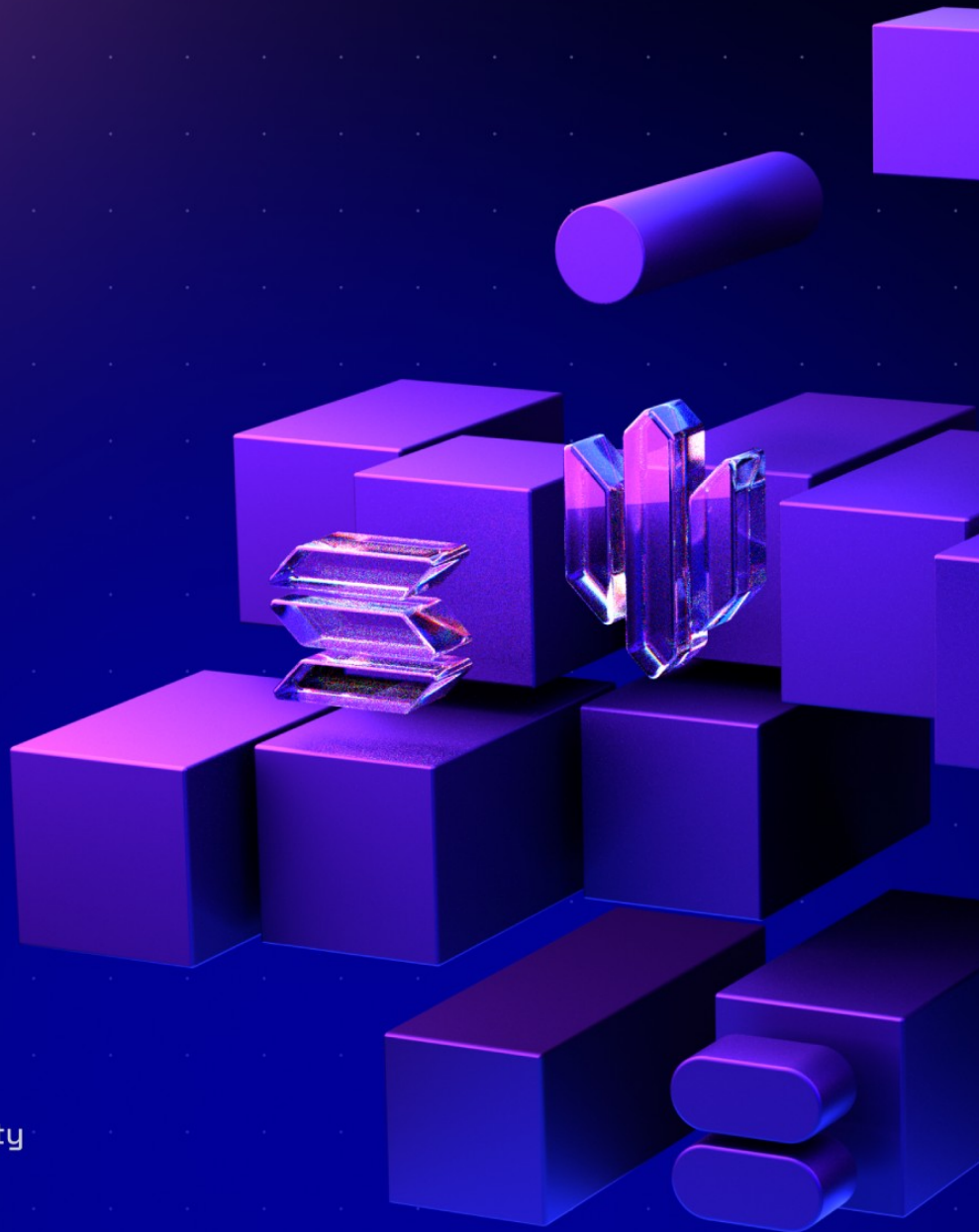


# Liquid Bitcoin

Lombard Finance

4.4.2025



# Contents

1. Document Revisions .....	4
2. Overview .....	5
2.1. Ackee Blockchain Security .....	5
2.2. Audit Methodology .....	6
2.3. Finding Classification .....	7
2.4. Review Team .....	9
2.5. Disclaimer .....	9
3. Executive Summary .....	10
Revision 1.0 .....	10
Revision 1.1 .....	11
Revision 2.0 .....	12
Revision 2.1 .....	13
4. Findings Summary .....	15
Report Revision 1.0 .....	18
Revision Team .....	18
System Overview .....	18
Trust Model .....	18
Findings .....	19
Report Revision 1.1 .....	56
Revision Team .....	56
System Overview .....	56
Trust Model .....	56
Report Revision 2.0 .....	57
Revision Team .....	57
System Overview .....	57
Trust Model .....	57

Findings ..... 58

Appendix A: How to cite ..... 67

# 1. Document Revisions

1.0-draft	Draft Report	18.03.2025
<a href="#">1.0</a>	Final Report	18.03.2025
<a href="#">1.1</a>	Fix Review	24.03.2025
2.0-draft	Draft Report	28.03.2025
<a href="#">2.0</a>	Final Report	28.03.2025
<a href="#">2.1</a>	Fix Review	04.04.2025

## 2. Overview

This document presents our findings in reviewed contracts.

### 2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling [Wake](#) for Ethereum and [Trident](#) for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the [School of Solana](#) and the [Solana Auditors Bootcamp](#).

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

#### **Ackee Blockchain a.s.**

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

<https://ackee.xyz>

[hello@ackee.xyz](mailto:hello@ackee.xyz)

## 2.2. Audit Methodology

The Ackee Blockchain Security auditing process follows a routine series of steps:

### 1. Code review

- a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
- b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:  
  
missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows.
- c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.
- d. Review of best practices to improve efficiency, clarity, and maintainability.

### 2. Testing and automated analysis

- a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework [Trident](#).

### 3. Local deployment + hacking

- a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are unique to its implementation.

## 2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

*Low* to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

### Severity

		<i>Likelihood</i>			
		High	Medium	Low	N/A
<i>Impact</i>	High	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

## Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

## Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.



## 2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member's Name	Position
Andrej Lukačovič	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## 2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

## 3. Executive Summary

Liquid Bitcoin is a protocol that allows users to obtain bridged Bitcoin in the form of Solana SPL Tokens (referred to as **LBTC**).

### Revision 1.0

Lombard Finance engaged Ackee Blockchain Security to perform a security review of the Liquid Bitcoin protocol with a total time donation of 12 engineering days in a period between March 3 and March 18, 2025, with Andrej Lukačovič as the lead auditor.

The audit was performed on the commit `9171ae4`<sup>[1]</sup> and the scope was the following:

- [Lombard Finance Solana Contracts](#), excluding external dependencies;

We began our review by familiarizing ourselves with the codebase and the business logic of the scope. A significant amount of time was spent reviewing the documentation and researching the broader scope of the protocol (e.g., Babylon Bitcoin staking).

After completing the initial research, we proceeded with the manual review of the codebase. The manual review consisted of multiple stages, with the first stage focusing on understanding the codebase in general:

- the components of the Solana program;
- all instructions the program accepts;
- the architecture and structure of the codebase; and
- all information the project stores on-chain.

After establishing this initial understanding, we moved forward with the second stage, where we performed a line-by-line code review. This stage

consisted of more in-depth analysis of the code, examining potential issues, bugs, and security concerns.

In the final stage of the audit, we performed proof of concepts and wrote the report.

During the manual review, we paid special attention to:

- ensuring the project is correctly initialized and configured;
- verifying the minting of **LBTC** is securely handled;
- confirming the validation process cannot be bypassed;
- ensuring the protocol behaves transparently and as expected;
- verifying there are no mechanisms which could be used against users; and
- looking for common issues which could occur in the codebase.

Our review resulted in 17 findings, ranging from Info to High severity.

The most severe finding [H1](#) allows unauthorized minting of **LBTC** tokens by bypassing signature validation when the initial validator set is not configured yet, as the initial value of `weights_threshold` is 0.

Ackee Blockchain Security recommends Lombard Finance to address all reported issues.

See [Report Revision 1.0](#) for the system overview and trust model.

## Revision 1.1

The review was done on the given commit `ca1ccb2`<sup>[2]</sup>.

The issue [M3](#) was acknowledged by the client, with plans to address the issue in the future.

The issue [W3](#) was acknowledged by the client. The protocol allows setting the

initial validator set with a preference for a particular validator. However, subsequent validator set updates follow secure practices, requiring consortium approval for modifications.

The issue [W7](#) was acknowledged by the client, who provided additional clarification in the issue description.

The issue [W8](#) was acknowledged by the client.

The issue [W9](#) was acknowledged by the client, as the `bascule` program was not developed at the time of the revision.

## Revision 2.0

Lombard Finance engaged Ackee Blockchain Security to perform a security review of the Liquid Bitcoin protocol with a total time donation of 3 engineering days in a period between March 25 and March 28, 2025, with Andrej Lukačovič as the lead auditor.

The audit was performed on the commit `c96dc36`<sup>[3]</sup> and the scope was the following:

- [Lombard Finance Solana Contracts](#), excluding external dependencies;
- [Bascule program](#), excluding external dependencies.

The revision began with a review of the new addition to the scope, the Bascule program. During the revision of the Bascule program, we tested that the protocol works as intended with writing proof of concept tests.

The review continued with a deeper understanding of the program, during which we ensured that:

- it is correctly used during the Cross-Program Invocation (CPI) from the LBTC program;

- only the appointed reporter can submit new deposits;
- only the appointed validator can validate the deposits;
- all potential scenarios are correctly covered (for example, scenarios where the deposit is under the threshold of validation); and
- all mint requests are still correctly validated and cannot be bypassed.

Our review resulted in 5 findings, ranging from Info to Medium severity. The most severe one [M5](#).

The issue is caused by using the Config account from the LBTC program as a validator for the Bascule program, `validate_withdrawal` instruction. In Bascule, the validator is marked as rent payer if the deposit account does not exist yet. In this case, the Config account would be the rent payer. However, data accounts cannot be rent payers, thus preventing the mint instruction from being executed successfully.

Ackee Blockchain Security recommends Lombard Finance to address all reported issues.

See [Report Revision 2.0](#) for the system overview and trust model.

## Revision 2.1

The review was performed on commit `9001c77`<sup>[4]</sup>.

The fix review was performed only on the fixes provided for the [Report Revision 2.0](#). The scope contained additions to the source code (e.g., `change_mint_auth`) that were not reviewed, as these additions were not in the scope during [Report Revision 2.0](#).

The issue [M5](#) was fixed by the client by introducing a new account which serves as a payer for the deposit account creation.

The issue [M6](#) was fixed by the client by marking the Deposit account as mutable.

The remaining issues were fixed by the client, as recommended in the recommendations section.

[1] full commit hash: [9171ae4f4b6506595e03fd3db9ff443067c79463](#)

[2] full commit hash: [ca1ccb22cb1a62769e416230bffb9b1f2334bf46](#)

[3] full commit hash: [c96dc36ba1df984e3a9d094a52d1a9a5167fdc24](#)

[4] full commit hash: [9001c77520c7441982971c0c70c30a609f2d81f8](#)

## 4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	1	6	1	11	3	22

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
<a href="#">H1: Possible unauthorized LBTC minting</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">M1: Possible inadequate fees</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">M2: Possible initialization front-running</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">M3: Redeem does not allow for assets refund</a>	Medium	<a href="#">1.0</a>	Acknowledged
<a href="#">M4: Minters are security hazard</a>	Medium	<a href="#">1.0</a>	Fixed

Finding title	Severity	Reported	Status
<a href="#">L1: Uniqueness of Role-based Access Control is not guaranteed</a>	Low	<a href="#">1.0</a>	Fixed
<a href="#">W1: Inability to transfer Config authority</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W2: Treasury might cause protocol not operational</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W3: Weighted validator signatures</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W4: Deprecated Cross Program Invocation call</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W5: Fields might be uninitialized</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W6: <del>UnstakeRequest</del> does not take fee into consideration</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W7: Potential panicking due to arithmetic overflow</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W8: Unexpected behavior in vector boundaries</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W9: Unfinished code may trigger undesired behavior</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">I1: Inaccurate comment</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I2: Code quality can be improved</a>	Info	<a href="#">1.0</a>	Fixed



Finding title	Severity	Reported	Status
<a href="#">M5: Inability to execute Cross Program Invocation due to Config account being Rent payer</a>	Medium	<a href="#">2.0</a>	Fixed
<a href="#">M6: Inability to execute Cross Program Invocation due to immutable account</a>	Medium	<a href="#">2.0</a>	Fixed
<a href="#">W10: Bascule Initialization front-running</a>	Warning	<a href="#">2.0</a>	Fixed
<a href="#">W11: Inability to transfer <u>BasculeData</u> authority</a>	Warning	<a href="#">2.0</a>	Fixed
<a href="#">I3: Unnecessary storage of the Bascule program in the Config account</a>	Info	<a href="#">2.0</a>	Fixed

Table 3. Table of Findings

# Report Revision 1.0

## Revision Team

Member's Name	Position
Andrej Lukačovič	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## System Overview

The Lombard's Liquid Bitcoin protocol is a Solana-based program written using the Anchor framework. The protocol enables users to obtain bridged Bitcoin in the form of Solana SPL Tokens (referred to as **LBTC**).

The protocol stores critical operational information in the Config Account.

Users can obtain the bridged version of Bitcoin after validators sign the transaction message which is validated against the stored set of validator addresses. The protocol implements role-based operations as follows:

- minters can mint and burn tokens;
- pausers can pause the protocol;
- claimers can execute `mint_with_fee` operation; and
- operators can set mint fees.

## Trust Model

Although the protocol implements Role-Based Access Control (RBAC) with multiple permission levels and message validation process is correctly implemented, users must trust:

- the Config admin to set appropriate operational fees;

- the Config admin to assign minters with security considerations, as minters can mint new tokens into circulation on the Solana blockchain (described in [M4](#));
- the protocol to maintain adequate validation, since the minimum limit for off-chain validators is set to 1 (enabling potential centralization, described in [W3](#)); and
- the protocol to correctly initialize the **LBTC** token, which means not misusing the `freeze_authority` or available Token-2022 extensions (this token information will be publicly available on the Solana blockchain and can be verified using explorers like [Solscan.io](#)).

## Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, [Go back to Findings Summary](#)

# H1: Possible unauthorized LBTC minting

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	validation.rs	Type:	Front-running

## Description

The `initialize` instruction initializes the Config account. The instruction sets the config admin and mint addresses while leaving the remaining fields set to their default values (zero or false).

The following code snippet shows the implementation of the `initialize` instruction in the `initialize.rs` file:

*Listing 1. Excerpt from initialize.rs*

```
1  #[derive(Accounts)]
2  pub struct Initialize<'info> {
3      #[account(mut)]
4      pub payer: Signer<'info>,
5      #[account(
6          init,
7          seeds = [b"lbtconfig"],
8          bump,
9          payer = payer,
10         space = 8 + Config::INIT_SPACE
11     )]
12     pub config: Account<'info, Config>,
13     pub system_program: Program<'info, System>,
14 }
15 pub fn initialize(
16     ctx: Context<Initialize>,
17     admin: Pubkey,
18     mint: Pubkey
19 ) -> Result<()> {
20     ctx.accounts.config.admin = admin;
21     ctx.accounts.config.mint = mint;
22     Ok(())
}
```

```
23 }
```

To mint new **LBTC** tokens, validation from validators is required. The weight of correct signatures must be equal to or higher than the `weight_threshold` configured within the Config account.

*Listing 2. Excerpt from validation.rs*

```
1 pub fn post_validate_mint<'info>(<br>2     config: &Account<'_, Config>,<br>3     recipient: &InterfaceAccount<'_, TokenAccount>,<br>4     mint_payload: &[u8],<br>5     weight: u64,<br>6     _bascule: &UncheckedAccount<'info>,<br>7 ) -> Result<u64> {<br>8     let mint_action = decoder::decode_mint_action(&mint_payload)?;<br>9<br>10    // ...<br>11    require!(<br>12        weight >= config.weight_threshold,<br>13        LBTCError::NotEnoughSignatures<br>14    );<br>15    // ...<br>16<br>17    Ok(mint_action.amount)<br>18 }
```

Since the numbers are not initialized to non-default values during the `initialize` instruction, the validation is vulnerable to front-running. The initial `weight_threshold` is set to 0, allowing the attacker to mint new **LBTC** tokens, without going through the signatures validation process.

## Exploit scenario

Alice is the deployer of the Solana program (the client's team).

Bob is an attacker.

1. Alice deploys the Solana program;

2. Bob creates `mint_payload` which contains his wallet as a recipient;
3. Bob invokes the `mint_from_payload` instruction, bypassing the required number of signatures check as `weight_threshold` is set to 0; and
4. the Solana program mints LBTC tokens to Bob's wallet.

## Recommendation

1. Ensure all values in the Config account are initialized.
2. Set `weight_threshold` to the type's maximum value; the threshold will be updated with the first set of validators.

## Fix 1.1

The issue was fixed by implementing a check that prevents the creation of mint payloads when the `weight_threshold` is set to zero.

[Go back to Findings Summary](#)

# M1: Possible inadequate fees

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	redeem.rs, mint_with_fee.rs	Type:	Data validation

## Description

The admin address stored within the Config account can update the following field:

- `burn_commission` - used in the `redeem` instruction.

The admin can also appoint an `operator`, who has authority over:

- `mint_fee` - used in the `mint_with_fee` instruction.

These fields determine the fees deducted during their corresponding instructions. The corresponding authorities can set these fields to any value, including potentially excessive amounts.

## Exploit scenario

Alice is a Config authority.

Alex is an operator.

Bob is a regular user.

1. Alice sets the `burn_commission` to an excessive amount;
2. Alex sets the `mint_fee` to an excessive amount;
3. Bob decides to redeem his LBTC for BTC;
4. Bob pays the unexpectedly high fee for the `redeem` instruction;

5. Bob decides to bridge the BTC back to LBTC;
6. BTC is bridged back using the `mint_with_fee` instruction; and
7. Bob pays the unexpectedly high fee for the `mint_with_fee` instruction.

## Recommendation

1. Replace the constant fee values (`mint_fee` and `burn_commission`) with basis points (100 basis points = 1%).
2. Implement limits on fee basis points within a reasonable range (e.g., 0-100).

## Fix 1.1

The issue was fixed by implementing a check that the `burn_commission` and `mint_fee` are smaller or equal to 100000.

[Go back to Findings Summary](#)



## M2: Possible initialization front-running

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	initialize.rs	Type:	Front-running

### Description

The `initialize` instruction initializes the Config account. The instruction sets the Config admin and mint address.

The following code snippet shows the implementation of the `initialize` instruction in the `initialize.rs` file:

*Listing 3. Excerpt from initialize.rs*

```
1  #[derive(Accounts)]
2  pub struct Initialize<'info> {
3      #[account(mut)]
4      pub payer: Signer<'info>,
5
6      #[account(
7          init,
8          seeds = [b"lbtconfig"],
9          bump,
10         payer = payer,
11         space = 8 + Config::INIT_SPACE
12     )]
13     pub config: Account<'info, Config>,
14     pub system_program: Program<'info, System>,
15 }
16
17 pub fn initialize(ctx: Context<Initialize>, admin: Pubkey, mint: Pubkey) ->
    Result<()> {
18     ctx.accounts.config.admin = admin;
19     ctx.accounts.config.mint = mint;
20     Ok(())
21 }
```

A vulnerability similar to [H1](#) exists. In this case, an attacker can front-run the `initialize` instruction, gaining authority over the Config account. This would allow the attacker to update the Config account and potentially mint new `LBTC` tokens.

## Exploit scenario

Alice is the deployer of the Solana program (the client's team).

Bob is an attacker.

1. Alice deploys the Solana program;
2. Bob front-runs the `initialize` instruction and invokes it with his wallet as admin and `LBTC` mint address;
3. Bob gains authority over the Config account;
4. Bob can set his wallet as authorized `minter`, which would allow him to mint new `LBTC` tokens; or
5. Bob can leave the validators and `weight_threshold` unset, which would allow him to process any `mint_payload`.

This attack requires that the mint is already initialized with the mint authority set to the Program Derived Address (PDA) expected by the program. Therefore, the likelihood is low.

## Recommendation

Ensure the `initialize` instruction can be invoked only by the upgrade authority of the Solana program.

### Fix 1.1

The issue was fixed by implementing the check that only the `upgrade_authority` can invoke the `initialize` instruction.

[Go back to Findings Summary](#)

## M3: Redeem does not allow for assets refund

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	redeem.rs	Type:	Trust model

### Description

The `redeem` instruction allows users to redeem their `LBTC` tokens and receive the original assets (on-chain Bitcoin). This instruction burns the user's funds and transfers some amount to the Treasury account as a fee without performing any validation or locking mechanism.

If the off-chain components fail, there is no guarantee that users will receive their assets back.

As discussed with the client:

*Ackee: What is the expected flow of the redeem instruction?*

*The instruction is meant to allow users to convert from LBTC to BTC on-chain, right?*

*Lombard: yes, burn lbtc on Solana, the consortium sign transfer of btc on Bitcoin*

*Ackee: What will be the steps if someone does not receive BTC? Will they be able to receive their LBTC back?*

*Lombard: the backend handle the failure situation and either retries or anyway let us handle the situation to refund the user.*

*Ackee: What does it mean "to refund the user" ? How long will*

*the refund take ?*

*Lombard: refund I mean complete the redeem by transferring the btc amount. I think we pay hanging unstakes on weekly or biweekly basis*

*Ackee: So it means if they execute the redeem, they will always receive the BTC. So a refund cannot happen on Solana with returning the LBTC, right?*

*Lombard: Correct*

— Lombard Finance Team

The redeem process relies on off-chain components for completion. While the likelihood is marked as Low, the potential for loss of funds remains present.

## Exploit scenario

Alice is the protocol operator (the client's team).

Bob is a regular user of the protocol.

1. Bob initiates a redeem of 1 **LBTC** token;
2. off-chain components are not operational;
3. Bob does not receive his 1 Bitcoin;
4. Bob cannot roll back the redeem instruction after a time period has passed; and
5. Bob has no guarantee that the redeem will be successful in the future.

## Recommendation

Implement an escrow-based refund mechanism for users who do not receive their BTC:

- lock funds in the Solana program for a defined period;
- allow off-chain components to complete the redeem during this period;
- return funds to the user if completion fails; and
- burn funds and transfer to treasury upon successful completion.

Implement a robust validation process. Require off-chain validators to verify Bitcoin network transfers.

### **Acknowledgment 1.1**

The issue was acknowledged by the client, with plans to implement an escrow based refund in the future.

[Go back to Findings Summary](#)

## M4: Minters are security hazard

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	mint.rs	Type:	Data validation

### Description

Minters are appointed by the Config admin.

The Config account contains a field that stores a list of minter addresses. These addresses have unrestricted permission to mint new **LBTC** tokens. Any address from this list can call the **mint** instruction, which will:

- mint new **LBTC** to the Token Account of the minter; and
- mint new **LBTC** without any additional validation.

### Exploit scenario

Alice is a minter appointed by the Config admin.

Bob is a malicious actor who has compromised Alice's wallet.

1. Bob gains access to Alice's private key through a hack or leak;
2. Bob executes the **mint** instruction to add new **LBTC** to circulation on the Solana blockchain; and
3. Bob executes the **redeem** instruction to redeem the newly minted **LBTC** for the original assets (on-chain Bitcoin).

### Recommendation

Either:

1. Remove the minter functionality completely; or
2. Add additional validation requirements to the `mint` instruction.

### Fix 1.1

The issue was fixed by removing the minter functionality. More precisely:

- The `Minter` role was removed;
- The `mint` instruction was removed; and
- The `burn` instruction was removed.

[Go back to Findings Summary](#)



# L1: Uniqueness of Role-based Access Control is not guaranteed

*Low severity issue*

Impact:	Medium	Likelihood:	Low
Target:	admin.rs	Type:	Access control

## Description

Minters, Claimers, and Pausers are additional roles appointed by the Config admin.

These roles have access to the following instructions:

- Minters: `mint` and `burn` instructions;
- Claimers: `mint_with_fee`; and
- Pausers: `pause`.

The roles are stored within lists (arrays) in the Config struct:

*Listing 4. Excerpt from state.rs*

```
1 #[account]
2 #[derive(InitSpace)]
3 pub struct Config {
4     // ...
5     #[max_len(10)]
6     pub minters: Vec<Pubkey>,
7     #[max_len(10)]
8     pub claimers: Vec<Pubkey>,
9     #[max_len(10)]
10    pub pausers: Vec<Pubkey>,
11    // ...
12 }
```

These containers do not ensure address uniqueness. Furthermore, the

instructions that allow the admin to remove an address only remove a single occurrence of that address. For example:

*Listing 5. Excerpt from admin.rs*

```
1 pub fn remove_minter(ctx: Context<Admin>, minter: Pubkey) -> Result<()> {
2     let mut found = false;
3     let mut index = 0;
4     for (i, m) in ctx.accounts.config.minters.iter().enumerate() {
5         if *m == minter {
6             found = true;
7             index = i;
8         }
9     }
10
11     // HACK: this will remove only one occurrence, not all
12     if found {
13         ctx.accounts.config.minters.swap_remove(index);
14         emit!(MinterRemoved { minter });
15     }
16     Ok(())
17 }
```

The code above shows that only the last found occurrence of the address is removed, leaving potential duplicates in the list.

## Exploit scenario

Alice is the Config admin.

Bob is a user who will be appointed as a minter.

1. Alice adds Bob as a minter;
2. Alice mistakenly adds Bob again as a minter;
3. Alice removes Bob from minters, believing the role was completely removed; and
4. Bob retains minter access due to the duplicate entry (i.e., Bob can still mint **LBTC** tokens).

## Recommendation

Either:

1. Implement uniqueness validation when adding addresses to role lists; or
2. Modify removal functions to remove all occurrences of an address from the role lists.

## Fix 1.1

The issue was fixed by keeping the addresses stored in the role lists unique.

[Go back to Findings Summary](#)

## W1: Inability to transfer Config authority

Impact:	Warning	Likelihood:	N/A
Target:	admin.rs	Type:	Access control

### Description

The Config admin is a critical role with authority to update most of the fields of the Config account, either directly or indirectly. However, there is no instruction that allows transferring the Config admin authority to another address. This functionality is essential in cases where the Config admin wallet is compromised.

### Recommendation

Implement a two-step authority transfer mechanism for the Config admin role:

1. Config admin appoints a new pending admin; and
2. pending admin accepts the transfer process.

### Fix 1.1

The issue was fixed by implementing a two-step authority transfer mechanism for the Config admin role.

[Go back to Findings Summary](#)

## W2: Treasury might cause protocol not operational

Impact:	Warning	Likelihood:	N/A
Target:	admin.rs	Type:	Data validation

### Description

Treasury is a field in the Config account appointed by the Config admin. The Treasury is responsible for collecting operational fees.

The `set_treasury` instruction allows the Config admin to appoint a new treasury. The following source code listing shows the `set_treasury` instruction logic:

*Listing 6. Excerpt from admin.rs*

```
1  #[derive(Accounts)]
2  pub struct Admin<'info> {
3      #[account(address = config.admin)]
4      pub payer: Signer<'info>,
5      #[account(mut)]
6      pub config: Account<'info, Config>,
7  }
8
9  pub fn set_treasury(ctx: Context<Admin>, treasury: Pubkey) -> Result<()> {
10     ctx.accounts.config.treasury = treasury;
11     emit!(TreasuryChanged { address: treasury });
12     Ok(())
13 }
```

The source code does not validate that the new treasury is a valid token account corresponding to the `LBTC` mint. If set incorrectly, the `mint_with_fee` and `redeem` instructions will become non-operational, as the fee transfer to the treasury will fail.

## Recommendation

Implement validation in the `set_treasury` instruction to verify that the new treasury is a valid token account corresponding to the `LBTC` mint.

### Fix 1.1

The issue was fixed by implementing a check that the treasury is a valid token account corresponding to the `LBTC` mint.

[Go back to Findings Summary](#)

## W3: Weighted validator signatures

Impact:	Warning	Likelihood:	N/A
Target:	post_mint_signatures.rs, post_valset_signatures.rs	Type:	Trust model

### Description

The Solana protocol uses validators to validate messages for:

- updating the set of validator addresses with their corresponding weights;  
and
- minting new **LBTC** tokens.

The validator set supports weighted signatures, allowing different weights for different validator signatures. The initial validator set is assigned by the Config Admin.

While new validator sets require sufficient signatures from the previous validator set for approval, the initial set of validators can be configured with unfair weight distribution.

### Exploit scenario

Consider a validator set with 100 validators:

Cluster A: - 90 validators with signature weight of 1 each - Total weight: 90

Cluster B: - 10 validators with signature weight of 20 each - Total weight: 200

With a **weight\_threshold** of 70:

1. Only 4 validators from Cluster B (weight: 80) can approve a message; while
2. 70 validators from Cluster A are needed for the same approval.

## **Recommendation**

Implement equal weighting for all validators (weight of 1 per signature).

## **Acknowledgment 1.1**

The issue is acknowledged by the client. The client is aware of the issue

[Go back to Findings Summary](#)



## W4: Deprecated Cross Program Invocation call

Impact:	Warning	Likelihood:	N/A
Target:	redeem.rs	Type:	Standards violation

### Description

The program uses the deprecated `anchor_spl::token_interface::transfer` invocation for token transfer in the `redeem` instruction.

### Recommendation

Use the `anchor_spl::token_interface::transfer_checked` invocation instead.

### Fix 1.1

The issue was fixed by replacing the deprecated `transfer` instruction with the `transfer_checked` instruction.

[Go back to Findings Summary](#)

## W5: Fields might be uninitialized

Impact:	Warning	Likelihood:	N/A
Target:	initialize.rs	Type:	Configuration

### Description

The Config account is initialized using the `initialize` instruction. This instruction sets only 2 out of 17 fields. Unset fields default to zero, false, or empty vectors. The following critical operational fields are affected:

- `withdrawals` (`withdrawals_enabled`) are disabled by default;
- `treasury` is set to default Pubkey (i.e. `11111111111111111111111111111111`);
- `burn_commission` is set to 0;
- `dust_fee_rate` is set to 0;
- `mint_fee` is set to 0; and
- `weight_threshold` is set to 0.

### Recommendation

1. Modify the `initialize` instruction to require and set all Config account fields.
2. Implement validation to ensure no critical fields remain at default values.

### Fix 1.1

Most of the fields are initialized as recommended. The following fields remain uninitialized:

- `withdrawals_enabled` is set to `false`
- `weight_threshold` is set to 0

The disabled withdrawals by default are justified, requiring manual activation by the Config admin. The `weight_threshold` initialization to 0 is addressed through improved validation that prevents bypassing the validation process.

The remaining fields in the Config account are initialized to default values, which does not impact the protocol's security or functionality.

[Go back to Findings Summary](#)

## W6: `UnstakeRequest` does not take fee into consideration

Impact:	Warning	Likelihood:	N/A
Target:	redeem.rs	Type:	Logic error

### Description

The `redeem` instruction allows users to redeem their `LBTC` tokens for Bitcoin. The instruction takes the desired amount to redeem and deducts a fee from this amount.

After the fee is transferred to the treasury and the remaining amount is burned, an `UnstakeRequest` event is emitted to inform the off-chain components about the redeem request.

*Listing 7. Excerpt from `redeem.rs`*

```
1 pub fn redeem(ctx: Context<Redeem>, script_pubkey: Vec<u8>, amount: u64) ->
  Result<()> {
2     // redeem instruction logic
3     // ...
4
5     emit!(UnstakeRequest {
6         from: ctx.accounts.payer.key(),
7         script_pubkey,
8         amount, // the whole amount is emitted
9     });
10 }
```

The emitted `UnstakeRequest` event contains the original amount without accounting for the deducted fee. If the off-chain component processes this event and transfers the full amount to the user on the Bitcoin network, a balance discrepancy occurs.

## Exploit scenario

Alice is an off-chain component.

Bob is a user requesting redemption of 100 `LBTC` tokens.

The `burn_commission` fee is 1 `LBTC`.

1. Bob requests the `redeem` instruction of 100 `LBTC` tokens;
2. the instruction transfers 1 `LBTC` as a fee to the treasury;
3. the instruction burns 99 `LBTC` tokens;
4. the `UnstakeRequest` event emits with 100 `LBTC` as the requested amount;
5. if the off-chain component does not account for the fee, it transfers 100 `LBTC` to the user; and
6. a balance discrepancy occurs, as the fee was deducted but the user received the full amount (effectively burning 99 `LBTC` but transferring 100 Bitcoin to the user).

## Recommendation

Either:

1. Include the fee amount in the `UnstakeRequest` event; or
2. Emit the actual amount to be transferred (post-fee amount) instead of the original request amount.

## Fix 1.1

The issue was fixed by emitting the actual amount to be transferred (`amount - fee`) instead of the original request amount.

[Go back to Findings Summary](#)

## W7: Potential panicking due to arithmetic overflow

Impact:	Warning	Likelihood:	N/A
Target:	bitcoin_utils.rs, validation.rs	Type:	Arithmetics

### Description

The Solana program uses unchecked arithmetic operations. The root `Cargo.toml` correctly sets:

*Listing 8. Excerpt from Cargo.toml*

```
1 [profile.release]
2 overflow-checks = true
```

which ensures overflow/underflow will cause a panic. However, using checked or saturating arithmetic is still recommended, as saturating arithmetic will return the type's maximum value instead of panicking.

Two critical locations in the code where unchecked arithmetic may prevent operation completion:

#### `get_dust_limit_for_output`

The function may overflow if `dust_fee_rate` from the Config account is too high. There are no restrictions on `dust_fee_rate` values.

*Listing 9. Excerpt from bitcoin\_utils.rs*

```
1 pub fn get_dust_limit_for_output(script_pubkey: &[u8], dust_fee_rate: u64) ->
   Result<u64> {
2     // ...
3     let spend_cost = BASE_SPEND_COST + WITNESS_INPUT_SIZE +
       serialize_size(script_pubkey_len);
4     // potentially overflow in multiplication
5     Ok((spend_cost * dust_fee_rate) / 1000)
```

```
6 }
```

### `validate_valset`

The function may overflow if the sum of validator weights exceeds `u64::MAX`.

*Listing 10. Excerpt from validation.rs*

```
1 pub fn validate_valset(  
2     validators: &[[u8; constants::VALIDATOR_PUBKEY_SIZE]],  
3     weights: &[u64],  
4     weight_threshold: u64,  
5 ) -> Result<()> {  
6     // ...  
7  
8     let mut sum = 0;  
9     for weight in weights {  
10         require!(*weight > 0, LBTError::ZeroWeight);  
11         sum += weight;  
12     }  
13  
14     // ...  
15 }
```

## Recommendation

1. Implement checked arithmetic operations in all arithmetic calculations; or
2. Use saturating arithmetic where overflow behavior is acceptable and maximum values are appropriate.

## Acknowledgment 1.1

The issue was acknowledged by the client, who provided the following explanation:

*Lombard: for the two outlined places, we find it quite hard to imagine a scenario in which this overflows 64 bits. in the calculation of dust fee, this is just an equation of bitcoin*

*network parameters and this should never exceed the maximum supply of bitcoin which comfortably fits in 64 bits. for the summed weight, if this overflows are probably doing something very wrong with weight distribution anyway, and i can't imagine a sane scenario in which we would have a sum of weights close to the limit of a 64 bit integer. in which case, it's probably better not to use saturating arithmetic and allow the program to panic as it likely means the data is malformed*

— Lombard Finance Team

The intentional panic during the `validate_valset` function execution is justified.

The `get_dust_limit_for_output` function remains a potential source of panics if the `dust_fee_rate` is set to a very high value. Since the rate is upgradeable by the Config admin, it is their responsibility to ensure the rate does not cause an overflow.

[Go back to Findings Summary](#)



## W8: Unexpected behavior in vector boundaries

Impact:	Warning	Likelihood:	N/A
Target:	state.rs, validation.rs	Type:	Logic error

### Description

The protocol's structures store vectors among other fields. Although the `#[max_len(<SIZE>)]` attribute is used, it does not enforce the final length of the corresponding vector.

The following structures are affected:

- `Config`
- `Metadata`

### `Config`

The current implementation allows setting more than the intended maximum of 10 minters. The `#[max_len(10)]` attribute only ensures sufficient allocation space for 10 `Pubkey` elements. During serialization, if the `claimers` and `pausers` vectors are empty, the `minters` vector can exceed 10 elements.

*Listing 11. Excerpt from state.rs*

```
1  #[account]
2  #[derive(InitSpace)]
3  pub struct Config {
4      // ...
5      #[max_len(10)]
6      pub minters: Vec<Pubkey>,
7      #[max_len(10)]
8      pub claimers: Vec<Pubkey>,
9      #[max_len(10)]
10     pub pausers: Vec<Pubkey>,
11     // ...
12 }
```

## Metadata

The `Metadata` account exhibits the same issue, as the `post_metadata_for_valset_payload` instruction does not verify that the vectors within the account maintain equal lengths.

*Listing 12. Excerpt from state.rs*

```
1 #[account]
2 #[derive(InitSpace)]
3 pub struct Metadata {
4     #[max_len(MAX_VALIDATOR_SET_SIZE)]
5     pub validators: Vec<u8; VALIDATOR_PUBKEY_SIZE>,
6     #[max_len(MAX_VALIDATOR_SET_SIZE)]
7     pub weights: Vec<u64>,
8 }
```

## Recommendation

1. Implement runtime checks to enforce vector length constraints
2. Add validation to ensure vectors cannot exceed their maximum defined lengths

## Acknowledgment 1.1

The warning was acknowledged by the client.

[Go back to Findings Summary](#)

## W9: Unfinished code may trigger undesired behavior

Impact:	Warning	Likelihood:	N/A
Target:	validation.rs	Type:	Code quality

### Description

The `post_validate_mint` function contains a branch that triggers when bascule is enabled in the Config account. As shown in the following excerpt, this branch is unimplemented (marked with `todo!()`). When bascule is enabled, the function will panic, preventing the successful execution of `mint_from_payload` and `mint_with_fee` instructions.

*Listing 13. Excerpt from validation.rs*

```
1 pub fn post_validate_mint<'info>(<br>2     config: &Account<'_, Config>,<br>3     recipient: &InterfaceAccount<'_, TokenAccount>,<br>4     mint_payload: &[u8],<br>5     weight: u64,<br>6     _bascule: &UncheckedAccount<'info>,<br>7 ) -> Result<u64> {<br>8     // ...<br>9     // Confirm deposit against bascule, if using.<br>10    if config.bascule_enabled {<br>11        // TODO<br>12        // This is empty for now, while Bascule<br>13        // is being implemented as a Solana program.<br>14        todo!();<br>15    }<br>16    // ...<br>17 }
```

### Recommendation

Either:

- implement the missing bascule validation logic; or
- remove the unimplemented code block to prevent runtime panics.

### **Acknowledgment 1.1**

The issue was acknowledged by the client.

### **Fix 2.0**

The issue was fixed by implementing the Bascule program and adding Cross Program Invocation functionality to the program, when bascule is enabled.

[Go back to Findings Summary](#)

## I1: Inaccurate comment

Impact:	Info	Likelihood:	N/A
Target:	bitcoin_utils.rs	Type:	Code quality

### Description

The following comment contains an inaccurate calculation:

*Listing 14. Excerpt from bitcoin\_utils.rs*

```
1 const BASE_SPEND_COST: u64 = 41; // 32 (txid) + 4 (vout) + 1 (scriptSig size)
   + 4 (nSequence) + 8 (amount)
```

According to the Bitcoin codebase ([reference](#)), the computation of `BASE_SPEND_COST` should be `32 + 4 + 1 + 4` (41 bytes), without the additional 8 bytes for amount.

### Recommendation

Update the comment to accurately reflect the Bitcoin codebase implementation:

*Listing 15. Excerpt from bitcoin\_utils.rs*

```
1 const BASE_SPEND_COST: u64 = 41; // 32 (txid) + 4 (vout) + 1 (scriptSig size)
   + 4 (nSequence)
```

### Fix 1.1

The comment was fixed as recommended.

[Go back to Findings Summary](#)

## I2: Code quality can be improved

Impact:	Info	Likelihood:	N/A
Target:	Contract.sol	Type:	Code quality

### Description

The `abi_encode` function contains repetitive code that could be consolidated into a single function. Additionally, the function uses multiple magic numbers that should be defined as constants.

The Validator set payload hash is redundantly passed to multiple instructions:

- `create_metadata_for_valset_payload;`
- `post_metadata_for_valset_payload;`
- `create_valset_payload;`
- `post_valset_signatures;`
- `set_initial_valset;` and
- `set_next_valset.`

This hash could be stored in the `Metadata` and `ValsetPayload` accounts to simplify instruction arguments.

The `admin.rs` file contains functions for managing the Config account fields that are upgradable by the Config admin. The logic for removing addresses (Minter/Claimer/Pauser) from lists is identical for each role. This logic can be consolidated into a single function.

### Recommendation

1. Consolidate duplicate code in the `abi_encode` function into reusable functions;

2. define magic numbers as named constants;
3. store the Validator set payload hash in the `Metadata` and `ValsetPayload` accounts instead of passing it as an argument; and
4. create a single function to handle removing addresses from lists.

### **Fix 1.1**

The issue was fixed as recommended.

[Go back to Findings Summary](#)

# Report Revision 1.1

## Revision Team

Revision team is the same as in [Report Revision 1.0](#).

## System Overview

The system overview remained unchanged since the previous revision.

## Trust Model

Changes to the trust model:

- minters are no longer part of the Role-Based Access Control (RBAC) scheme;
- fees are upgradeable by the Config admin and are capped at reasonable values;

Remaining trust assumptions, from previous revision are still valid.



# Report Revision 2.0

## Revision Team

Member's Name	Position
Andrej Lukačovič	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## System Overview

The Lombard's Liquid Bitcoin protocol has been extended with a new program called Bascule. Bascule is a Solana program written using the Anchor framework that adds an additional layer of security to the protocol. The Bascule program validates requested deposits through a role called reporter. The reporter is an off-chain component responsible for posting requested deposits to the Solana blockchain. The LBTC program then compares the requested mints against the deposits to ensure the deposit was actually performed and reported by the reporter.

## Trust Model

Both programs implement a Role-Based Access Control (RBAC) mechanism. Users must trust:

- the Bascule admin to appoint a responsible reporter;
- the Bascule admin to appoint a responsible pauser (the pauser role is responsible for pausing the protocol);
- the Bascule admin to responsibly set the `validate_threshold` (deposits under this threshold do not need to be reported by the reporter); and
- all points from the previous revisions.

## Findings

The following section presents the list of findings discovered in this revision.  
For the complete list of all findings, [Go back to Findings Summary](#)

## M5: Inability to execute Cross Program Invocation due to Config account being Rent payer

*Medium severity issue*

Impact:	Medium	Likelihood:	Medium
Target:	mint_from_payload.rs, mint_with_fee.rs, validator.rs	Type:	Denial of service

### Description

Basculc contains a `validate_threshold` parameter. The threshold determines whether a deposit requires reporting:

- deposits below the threshold do not need to be reported by the reporter
- deposits greater than or equal to the threshold must be reported by the reporter

The issue occurs for deposits below the threshold.

In this case, the deposit account can be initialized by the validator instead of the reporter. The LBTC program in the Basculc Cross Program Invocation uses its Config Account as the validator. The validator is responsible for paying the rent fee for the Deposit account creation. However, data accounts cannot be rent payers, as they cannot have their balance subtracted using the system program transfer instructions.

### Exploit scenario

Alice is a regular user of the protocol who wants to bridge an amount smaller than the `validate_threshold`. The following scenario occurs:

1. Alice initiates a deposit;

2. Alice invokes the `mint_from_payload` instruction;
3. the validator (Config Account) attempts to pay the rent fee for the Deposit account creation;
4. the transaction fails because the Config Account is a data account and cannot be used as a rent payer; and
5. the execution results in a Denial of Service as the instruction cannot be completed.

## Recommendation

Modify the program to use a dedicated rent-paying account for deposit creation.

## Fix 2.1

The issue was fixed by adding new account which is responsible for paying the rent fee for the Deposit account creation.

[Go back to Findings Summary](#)

## M6: Inability to execute Cross Program Invocation due to immutable account

*Medium severity issue*

Impact:	Medium	Likelihood:	Medium
Target:	<code>mint_from_payload.rs</code> , <code>mint_with_fee.rs</code>	Type:	Denial of service

### Description

When the Bascule validation is invoked using the Cross Program Invocation, the Deposit account must be marked as mutable in the LBTC program's `mint_from_payload` and `mint_with_fee` instructions as it is going to be modified in the Bascule program. Currently, the account is not marked as mutable, causing the generated IDL to mark the accounts as read-only. This results in the `mint_from_payload` and `mint_with_fee` instructions failing, in case of bascule being activated.

### Exploit scenario

Alice is a regular user of the protocol who wants to bridge an amount smaller than the `validate_threshold`. The following scenario occurs:

1. Alice initiates a deposit;
2. Alice invokes the `mint_from_payload` instruction;
3. the Deposit account is not marked as mutable;
4. the Bascule Cross-Program Invocation fails due to writable privilege escalation; and
5. the execution results in a Denial of Service as the instruction cannot be completed.

## Recommendation

Mark the Deposit account as mutable in the LBTC program's `mint_from_payload` and `mint_with_fee` instructions.

### Fix 2.1

The issue was fixed by marking the Deposit account as mutable in the LBTC program's `mint_from_payload` and `mint_with_fee` instructions.

[Go back to Findings Summary](#)

## W10: Bascule Initialization front-running

Impact:	Warning	Likelihood:	N/A
Target:	bascule/initialize.rs	Type:	Front-running

### Description

The Bascule program is vulnerable to front-running attacks due to not limiting the payer account that is assigned as the admin.

### Recommendation

1. limit the Initialize instruction execution to only the `upgrade_authority` of the Bascule program; and
2. implement an admin role two-step transfer mechanism.

### Fix 2.1

The issue was fixed by limiting the Initialize instruction execution to only the `upgrade_authority` of the Bascule program.

[Go back to Findings Summary](#)

## W11: Inability to transfer **BasculeData** authority

Impact:	Warning	Likelihood:	N/A
Target:	bascule/lib.rs	Type:	Logic error

### Description

The Bascule program does not allow for the admin role of the BasculeData to be transferred. This is crucial in scenarios where the admin role is compromised.

### Recommendation

Implement a two-step transfer mechanism for the admin role.

### Fix 2.1

The issue was fixed by implementing a two-step transfer mechanism for the admin role.

[Go back to Findings Summary](#)



## I3: Unnecessary storage of the Bascule program in the Config account

Impact:	Info	Likelihood:	N/A
Target:	lbtc/state.rs	Type:	Code quality

### Description

The LBTC program's main Config account unnecessarily stores the Bascule program address. The field can be removed, and the correctness of the address can be validated within each particular instruction context.

Additionally, customization of the Bascule program can lead to undesired behavior if the Config admin's private key is compromised.

### Recommendation

Instead of storing the Bascule program address in the Config account, validate the address within each particular instruction context:

*Listing 16. Excerpt from validation.rs*

```
1 use bascule::program::Bascule;
2
3 pub struct MintWithFee<'info> {
4     // ...
5     pub bascule: Option<Program<'info, Bascule>>,
6     // ...
7 }
```

As the Bascule program is optional, wrap the account type in the `Option` type.

### Fix 2.1

The issue was fixed. However, the changes introduced a new issue that was subsequently fixed in commit [f287935](#)<sup>[1]</sup>.

[Go back to Findings Summary](#)

[1] full commit hash: `f2879353b8925e5437d95086803c8ce4db8a6230`

# Appendix A: How to cite

Please cite this document as:

[Ackee Blockchain Security](#), Liquid Bitcoin: Lombard Finance, 4.4.2025.



# Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4  
186 00 Prague  
Czech Republic

[hello@ackee.xyz](mailto:hello@ackee.xyz)