Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

LBTC Solana Program

Veridise Inc.

March 31, 2025

► **Prepared For:**

Lombard
https://www.lombard.finance/

► **Prepared By:**

Alberto Gonzalez
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Mar. 31, 2025 V1

Contents

1 Executive Summary

From Mar. 12, 2025 to Mar. 21, 2025, Lombard engaged Veridise to conduct a security assessment of their LBTC Solana Program. Previously, Veridise audited the EVM (Solidity) version of the LBTC contract*. This engagement focused on reviewing its Solana counterpart. While the Solana implementation preserves the intended functionality of the EVM version, it has been restructured to align with Solana's distinct development model and constraints.

Following the initial review, the developers requested an additional assessment of the newly completed Bascule feature. This supplementary review was conducted from Mar. 26, 2025 to Mar. 27, 2025.

Veridise conducted the assessment over 4 person-weeks, with 2 security analysts reviewing the project over 2 weeks. The LBTC program review was performed on commit `6fb4e11`, while the Bascule program review was conducted on commit `cc1efcc`.

The review strategy involved a tool-assisted analysis of the program source code as well as thorough code review performed by Veridise security analysts.

Project Summary. LBTC is a cross-chain, liquid, token that is backed 1:1 by Bitcoin. To obtain LBTC tokens, users send their native BTC tokens to a Lombard-controlled address. Once the tokens have been sent and processed, their LBTC counterpart is minted on the chosen destination chain in a 1:1 ratio. If necessary, users can then redeem their LBTC at any time to receive their BTC tokens (minus a fee) back.

The protocol behind LBTC consists of the following components:

- **Lombard Security Consortium** (LSC) - A group of trusted validator nodes that validates LBTC minting and redemption through a notarization proccess, ensuring only authorized actions are executed.
- **LBTC smart-contracts** - A set of programs on a blockchain responsible for minting and redeeming LBTC on request. A minting request must be signed by a sufficient number of LSC nodes.
- **Bascule** - An optional security layer for LBTC minting that safeguards against a compromised consortium. If enabled by the LBTC admin, it assigns an independent deposit reporter to authorize valid deposits. During minting, the system verifies deposits against the Bascule contract, rejecting any that lack approval, providing an added layer of security.

The LBTC Solana Program is the LBTC smart-contract designed to be executed on the Solana blockchain.

* The previous audit report, if publicly available, can be found on Veridise's website at `https://veridise.com/audits-archive/`

Code Assessment. The LBTC Solana Program developers provided the source code of the LBTC Solana Program contracts for the code review. The source code appears to be mostly original code written by the LBTC Solana Program developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise security analysts understanding of the code, the LBTC Solana Program developers explained the main concepts and workflows behind the provided code.

The source code included a comprehensive test suite that covered most of the protocol's workflows and tested both positive and negative scenarios.

Summary of Issues Detected. The security assessment uncovered 8 issues, 1 of which is assessed to be of high severity. Specifically, V-LBTC-VUL-001 would lead to sending an incorrect amount of BTC tokens to a user's wallet as a result of the LBTC redemption operation. The Veridise analysts also identified 2 low-severity issues, 3 warnings, and 2 informational findings. For example, the V-LBTC-VUL-002 would allow a malicious validator to gain an advantage over other validators during the voting process, which could lead to funds being drained in the worst-case scenario. Additionally, V-LBTC-VUL-003 would prevent the minting of LBTC tokens below the bascule threshold.

The LBTC Solana Program developers provided fixes for all of the findings.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the LBTC Solana Program.

Improve tests. While the test coverage and selection of test vectors appear to be generally well-chosen, the tests do not always accurately reflect real-world scenarios. For instance, the issue V-LBTC-VUL-001 could have been identified if the corresponding test had included verification of the content of emitted events. Additionally, issue V-LBTC-VUL-003, could have been identified through a test that included deposits below the bascule threshold.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
LBTC Solana Program	`6fb4e11`	Rust	Solana
Bascule Program	cc1efcc	Rust	Solana

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 12–Mar. 21, 2025	Manual	2	16 person-days
Mar. 26-Mar. 27, 2025	Manual	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	0	0	0
Low-Severity Issues	2	2	2
Warning-Severity Issues	3	3	3
Informational-Severity Issues	2	2	2
TOTAL	8	8	8

Table 2.4: Category Breakdown.

Name	Number
Logic Error	2
Access Control	2
Maintainability	2
Missing/Incorrect Events	1
Data Validation	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The Security Assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the code base for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

Additionally, several critical components of the codebase underwent rigorous fuzz testing. For further details, refer to Section 6.

3.2 Security Assessment Goals

The engagement was scoped to provide a security assessment of LBTC Solana Program's smart contracts.

During the assessment, the security analysts aimed to answer questions such as:

- Is the protocol free from access control vulnerabilities?
- Is the protocol free from common Solana program vulnerabilities?
- Does the logic of the protocol correspond to the reference EVM implementation?
- Are the protocol's logs emitted correctly?
- Are users able to mint LBTC tokens if and only if they have the corresponding validator set signatures?
- Is it possible to replay mint payloads?
- Is there a way to replace legitimate validator addresses with malicious ones?
- Is there any deficiency within the payload encoding and decoding functions that could give an advantage to an attacker?
- Is it possible for users to bypass the Bascule deposit validation?

3.3 Scope

The scope of this security assessment was limited to the below folders of the source code provided by the LBTC Solana Program developers:

1. `lbtc-solana-program/programs/lbtc/*` on commit `6fb4e11`
2. `lbtc-solana-program/programs/bascule/*` on commit `cc1efcc`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Trust Model

4.1 Operational Assumptions.

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for the LBTC Solana Program:

- **Validator Set**
  - The validator set consists only of honest participants who will not collude or act maliciously.
  - Validators will not sign fraudulent payloads that could lead to the minting of unbacked LBTC.
- **Bascule**
  - The Bascule deposit reporter is honest and consistently reports deposits in a timely manner.
- **Frontend**
  - The frontend does not introduce misleading information that could result in users signing unrealistic fee payloads expiries.

4.2 Privileged Roles.

Roles. This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensititivy. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive roles may be required to perform actions quickly based on real-time monitoring, while non-time-sensitive roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assume that the role operators perform their responsiblities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Highly-privileged, time-sensitive roles:
  - The LBTC admin can grant or remove the minter, pauser and claimer roles.
- Highly-privileged, non-time-sensitive roles:
  - The upgrade authorities, assigned to the LBTC and Bascule deployers, have the privilege to modify the protocol's programs code, effectively changing the programs logic at will.
  - The LBTC admin can modify the protocol configuration such as the bascule account, treasury account, dust fee rate, burn comission, enable and disable LBTC redemptions.

    - The LBTC minter role has the ability to mint tokens without signatures from the validator set.
- Limited-authority, time-sensitive roles:
    - The LBTC and Bascule pauser roles have the ability to pause and unpause the LBTC and Bascule operations, respectively.
    - The Bascule admin has the ability to change the deposit threshold.
    - The Bascule deposit reporter has the ability to approve deposits without needing to pass the threshold validation.
- Limited-authority, non-time-sensitive roles:
    - The LBTC operator role has the ability to change the mint fee
    - The LBTC claim role has the ability to mint tokens on another users's behalf with enough validator set signatures and a user's signature while claiming a fee from the minted tokens.

Operational Recommendations. The Veridise team recommends separating the the program deployers (upgrade authorities) from the role of protocol administrators. Specifically, the upgrade authorities for the LBTC and Bascule programs should not be assigned as the admins responsible for managing the LBTC and Bascule operations. This separation ensures that the keys controlling program upgrades are not used frequently for routine protocol management.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-LBTC-VUL-001	Incorrect amount emitted in . . .	High	Fixed
V-LBTC-VUL-002	Validators' set isn't enforced to contain . . .	Low	Fixed
V-LBTC-VUL-003	Bascule validation will not work with small . . .	Low	Fixed
V-LBTC-VUL-004	Function get_dust_limit_for_output . . .	Warning	Fixed
V-LBTC-VUL-005	Role discrepancy in unpause function	Warning	Fixed
V-LBTC-VUL-006	Lack of admin transfer mechanism	Warning	Fixed
V-LBTC-VUL-007	Maintainability issues	Info	Fixed
V-LBTC-VUL-008	Use of deprecated transfer method . . .	Info	Fixed

5.1 Detailed Description of Issues

5.1.1 V-LBTC-VUL-001: Incorrect amount emitted in UnstakeRequest event

Severity	High	**Commit**	6fb4e11
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`programs/lbtc/src/instructions/redeem.rs`		
Location(s)	redeem()		
Confirmed Fix At	450921a		

The `redeem()` function is responsible for handling the redemption of LBTC within the application. This function involves several key components: the `payer`, who signs the transaction; the `holder`, which is the account holding the LBTC; the `treasury`, where fees are collected; and the `mint`, which represents the LBTC token in the Solana Token Program. The function checks for certain conditions, such as whether withdrawals are enabled and if the amount is above the Bitcoin dust limit, before proceeding with the fee transfer and burn of the redeemed LBTC.

The issue arises in the event emission at the end of the `redeem()` function. The `UnstakeRequest` event is emitted with the `amount` parameter, which represents the total amount being redeemed. However, this does not account for the `fee` that is deducted during the process. As a result, the event inaccurately reflects the amount that is actually being redeemed, which should be `amount - fee`.

Impact This discrepancy is relevant because the off-chain component of the application relies on the `UnstakeRequest` event to determine how much LBTC was redeemed. This information is important for unlocking the equivalent amount in the Bitcoin network for the user. If the event emits the incorrect amount, it could lead to the off-chain component unlocking more Bitcoin than intended, resulting in potential financial discrepancies.

Recommendation To ensure the `UnstakeRequest` event accurately reflects the amount transferred to the user, modify the event emission to use `amount - fee` instead of `amount`.

Developer Response The developers implemented the suggested fix.

5.1.2 V-LBTC-VUL-002: Validators' set isn't enforced to contain unique elements

Severity	Low	**Commit**	6fb4e11
Type	Data Validation	**Status**	Fixed
File(s)	programs/lbtc/src/instructions/set_initial_valset.rs, programs/lbtc/src/instructions/post_metadata_for_valset_payload.rs		
Location(s)	set_initial_valset() ,post_metadata_for_valset_payload()		
Confirmed Fix At	322bffa		

The `set_initial_valset()` function and the `post_metadata_for_valset_payload()` function is used to either set or update the list of validators. Validators are users who are responsible for verifying token withdrawal requests by signing them. Signatures have varying weights, and their voting power is not equal. Once a certain threshold is reached in terms of the combined voting power, the request is executed.

```
pub fn post_metadata_for_valset_payload(
   ctx: Context<ValsetMetadata>,
   hash: [u8; 32],
   validators: Vec<[u8; VALIDATOR_PUBKEY_SIZE]>,
   weights: Vec<u64>,
) -> Result<()> {
   ctx.accounts.metadata.validators.extend(validators.clone());
   ctx.accounts.metadata.weights.extend(weights.clone());
   emit!(ValsetMetadataPosted { hash, validators, weights });
   Ok(())
}
```

Snippet 5.1: Snippet from `post_metadata_for_valset_payload()`

Please note that the provided validators list isn't enforced to contain unique addresses.

Impact Depending on the trust assumptions used in the protocol, a malicious validator could submit a new list where their address or some other address they control is repeated several times. This would give them an unfair advantage during voting, since they can provide their signature multiple times. In the worst case, this could lead to the treasury being drained.

Recommendation It is recommended to enforce uniqueness of address in the validators set collection.

Developer Response The developers implemented the suggested fix.

5.1.3 V-LBTC-VUL-003: Bascule validation will not work with small deposits

Severity	Low	**Commit**	cc1efcc
Type	Logic Error	**Status**	Fixed
File(s)	programs/bascule/src/instructions/validator.rs		
Location(s)	validate_withdrawal()		
Confirmed Fix At	85894ba		

The `validate_withdrawal()` function in the `Bascule` program is responsible for validating deposits in the `mint` flow of the `LBTC` program. Each deposit is represented as a separate PDA account, which can be in either the `Reported` state - created by the deposit reporter - or the `Unreported` state, which means it is not yet created.

For deposits below the `validate_threshold` - a configurable parameter - the deposit reporter does not need to pre-create the deposit account. Instead, the `validate_withdrawal()` function can create it on demand using the Anchor framework's `init_if_needed` feature:

```
#[account(
    // create the account if it doesn't already exist
    init_if_needed,
    // the validator pays for the account if it doesn't already exist
    payer = validator,
    // the PDA of the account: string "deposit" + the deposit id bytes
    seeds = [DEPOSIT_SEED, deposit_id.as_ref()], bump,
    // the fixed space for the account
    space = 8 + Deposit::INIT_SPACE,
)]
```

Snippet 5.2: Code snippet from the struct `Validator` of the `validator.rs` file.

Here, the `validator` account is designated as the payer for initialization. However, the current implementation assigns `config` (a PDA) as the `validator`, which is not owned by the `system_program`. Since only accounts controlled by `system_program` can pay for new account initialization, this setup causes transaction failures:

```
CpiContext::new_with_signer(
       bascule.to_account_info(),
       Validator {
         // @audit config account will act as the initialization payer for the
   deposit
         //.       account.
         validator: config.to_account_info(),
           bascule_data: bascule_data.to_account_info(),
           deposit: deposit.to_account_info(),
           system_program: system_program.to_account_info(),
       },
       .........
```

Snippet 5.3: Code snippet of the construction of the Validator context from the `validation.rs` file of the `LBTC` program codebase.

Impact Because the `LBTC.config` account cannot pay for deposit account initialization, any

deposit below the `validate_threshold` will fail. This results in a denial of service for smaller deposits, preventing users from successfully minting `LBTC` when the Bascule is enabled.

Recommendation To resolve this issue, the implementation should designate a different account-such as the transaction signer or another system-owned account-to act as the payer for the `deposit` account initialization.

Developer Response The developers implemented the suggested fix.

5.1.4 V-LBTC-VUL-004: Function get_dust_limit_for_output rounds result down instead of up

Severity	Warning	**Commit**	6fb4e11
Type	Logic Error	**Status**	Fixed
File(s)	`programs/lbtc/src/utils/bitcoin_utils.rs`		
Location(s)	get_dust_limit_for_output()		
Confirmed Fix At	1dffddf		

The function `get_dust_limit_for_output()` determines the minimum amount of BTC that a user can redeem in order to avoid the BTC dust problem.

Within the function, the expression incorrectly uses the division operator, rounding the result down instead of up.

```
/// Compute the dust limit for an output depending on its script pubkey type.
pub fn get_dust_limit_for_output(script_pubkey: &[u8], dust_fee_rate: u64) -> Result<
    u64> {
  let script_pubkey_len = script_pubkey.len();    // Validate correct output type,
    but we can drop the actual result.
  let _ = get_output_type(script_pubkey)?;
  let spend_cost = BASE_SPEND_COST + WITNESS_INPUT_SIZE + serialize_size(
    script_pubkey_len);
  Ok((spend_cost * dust_fee_rate) / 1000)
}
```

Snippet 5.4: Snippet from `get_dust_limit_for_output()`

Impact If a user redeems a value that is close to the dust limit, their redemption tokens may be locked forever, as it would not be possible to transfer them.

Recommendation It is recommended to implement the correct rounding behavior for the division operation.

Developer Response The developers implemented the suggested fix.

5.1.5 V-LBTC-VUL-005: Role discrepancy in unpause function

Severity	Warning	**Commit**	6fb4e11
Type	Access Control	**Status**	Fixed
File(s)	`programs/lbtc/src/lib.rs`		
Location(s)	unpause()		
Confirmed Fix At	048c3f4		

In the current implementation of the LBTC program on Solana, the `unpause()` function is controlled by the admin registered in the configuration account. This differs from the EVM version of the protocol, where the `unpause()` function is controlled by the Pauser role.

Impact This discrepancy in role-based access control could lead to inconsistencies in the management of the protocol's operational state.

Recommendation Use the `Pause` context instead of the `Admin` context.

Developer Response The developers implemented the suggested fix.

5.1.6 V-LBTC-VUL-006: Lack of admin transfer mechanism

Severity	Warning	**Commit**	6fb4e11
Type	Access Control	**Status**	Fixed
File(s)	`programs/lbtc/src/libr.rs`		
Location(s)	See description		
Confirmed Fix At	9ef3649		

The current implementation of the LBTC Solana program lacks a mechanism to change the admin. In contrast, the EVM version of the protocol implements a two-step ownership transfer process, which is absent in this implementation.

Impact The inability to change the admin restricts the protocol's flexibility and adaptability.

Recommendation Implement a two-step ownership transfer process similar to the EVM version.

Developer Response The developers implemented the suggested fix.

5.1.7 V-LBTC-VUL-007: Maintainability issues

Severity	Info	**Commit**	6fb4e11
Type	Maintainability	**Status**	Fixed
File(s)	`programs/lbtc/src/events.rs` , `programs/lbtc/src/errors.rs`		
Location(s)	See description		
Confirmed Fix At	19027f0		

The following constants from the `LBTCError` enum are not used anywhere:

- `Secp256k1RecoverError`
- `InvalidTreasury`
- `SignatureLengthMismatch`

The following events from the `events.rs` module are not used anywhere:

- `FeeActionSet`
- `BasculeAddressChanged`
- `ChainIdSet`
- `DepositBtcActionSet`
- `ValsetActionSet`

Impact The presence of unused constants and events can indicate either missing functionality or unnecessary code. Missing functionality can prevent the system from behaving as expected, while dead code can complicate the understanding of the code for developers.

Recommendation It is recommended to implement the missing functionality or remove unnecessary code from the codebase.

Developer Response The developers implemented the suggested fix.

5.1.8 V-LBTC-VUL-008: Use of deprecated transfer method in redeem function

Severity	Info	**Commit**	6fb4e11
Type	Maintainability	**Status**	Fixed
File(s)	`programs/lbtc/src/instructions/redeem.rs`		
Location(s)	redeem()		
Confirmed Fix At	25a80c8		

The `redeem()` function uses the `anchor_spl::token_interface::transfer` function to transfer tokens. However, in Token-2022, the transfer method has been deprecated and is recommended to use `transfer_checked()` instead.

Reference: https://docs.rs/spl-token-2022/latest/spl_token_2022/instruction/fn.transfer.html

Impact There is no direct impact, but using deprecated functionality is not a good practice.

Recommendation Replace the use of `transfer` with `transfer_checked` which has 2 additional parameters: `mint_pubkey` and `decimals`.

Developer Response The developers implemented the suggested fix.

6 Fuzz Testing

6.1 Methodology

The LBTC Solana Program implements several functions that are essential for the correct execution of the protocol. Ensuring the consistent and reliable behavior of these functions was of utmost importance. In particular, we focused on the following function:

1. The `ValSetAction::abi_encode()` function which is responsible for encoding the content of the structure `ValSetAction` according to Ethereum ABI-encode rules.
2. The `decoder::decode_mint_action()` function which is responsible for decoding the mint action from the provided payload.
3. The `decoder::decode_fee_action()` function which is responsible for decoding fee action from the provided payload.

The `ValSetAction::abi_encode()` function is responsible for encoding multiple arbitrarily sized collections in accordance with Ethereum ABI specifications, a process that we consider inherently complex and potentially prone to subtle bugs. Following a comprehensive manual code review, we opted to conduct fuzz testing on this function, using the canonical implementation provided by the `Ethers.rs` Rust package as a reference.

The functions `decoder::decode_mint_action()` and `decoder::decode_fee_action()` are relatively straightforward. However, it was essential to verify that no valid payload could trigger a panic within the Solana Program, as such an occurrence could result in a Denial of Service vulnerability.

6.2 Properties Fuzzed

Here, we express properties that we checked through fuzz testing.

- For all $x \in ValsetAction$, $x.abi\_encode() = encode(x)$, where $encode(x)$ denotes a reference implementation of the ABI-encode function.
- For all $x \in MintPayload$, $decode\_mint\_action(x) = Ok(\_)$
- For all $x \in MintPayload$, $decode\_fee\_action(x) = Ok(\_)$

After extensive fuzz testing, during which hundreds of millions of inputs were processed, no bugs were identified.