ClarityAlliance

ZEST PROTOCOL BTCz SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ABA

SEPTEMBER 20TH, 2024

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

2. Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

3. Introduction

A time-boxed security review of the Zest Protocol BTCz implementation, where Clarity Alliance reviewed the scope, whilst simultaneously building out a testing suite for the protocol.

4. About Zest Protocol

Zest Protocol is a decentralized lending platform on Stacks that enables users to trustlessly lend and borrow assets.

BTCz is a yield bearing BTC on Stacks. BTCz brings the best of Stacks and Babylon together to create the most secure BTC staking pool.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

CONTENTS

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.

- Medium - only a conditionally incentivized attack vector, but still relatively likely.

- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

6. Security Assessment Summary

Review Commit Hash:
f730701455a8d34444f9b95869bf6290f0a99112

- `stacking-btc.clar`
- `stacking-data.clar`
- `btc-registry.clar`
- `fee-data.clar`

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

7. Executive Summary

Over the course of the security review, Kristian Apostolov, ABA engaged with Zest Protocol to review Zest Protocol. In this period of time a total of **35** issues were uncovered.

Protocol Summary

Protocol Name	Zest Protocol
Date	September 20th, 2024

Findings Count

Severity	Amount
Low	7
QA	28
Total Findings	**35**

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

Summary of Findings

ID	Title	Severity	Status
[L-01]	Do not use tx-sender for sensitive operations	Low	Resolved
[L-02]	BTCz Token Name, Decimals, and Symbol Should Not Be Changeable	Low	Resolved
[L-03]	Fee Value Can Surpass 100% and Block Operations	Low	Resolved
[L-04]	Reward Commission Can Be Set Over 100% and Block Adding Rewards	Low	Resolved
[L-05]	Guard against withdrawal-direct-to-deposit edge-case	Low	Acknowledged
[L-06]	Bitcoin Deposits via PUSHDATA2 and PUSHDATA4 Opcodes Are Stuck	Low	Acknowledged
[L-07]	get-redeemable-btc-by-amount and get-redeemable-btc return misleading amount	Low	Resolved
[QA-01]	Deposits to Contracts with Long Names Are Stuck	QA	Acknowledged
[QA-02]	Redundant Fee Address Logic on Stacks	QA	Resolved
[QA-03]	Consider Moving Commission Logic Off-Chain	QA	Resolved
[QA-04]	Pausing Pegging In/Out Functionality Should Not Be in the Fee-Data Contract	QA	Resolved
[QA-05]	Revoking Withdrawals Not Implemented	QA	Resolved
[QA-06]	Overlapping Error Code Ranges	QA	Resolved
[QA-07]	Incorrectly Referencing BTCz as sBTC in Code	QA	Resolved
[QA-08]	Add Fee to stacking-btc::deposit Print Call for Better Off-Chain Tracking	QA	Resolved
[QA-09]	Misleading Amount Argument Name in stacking-btc::get-redeemable-btc-by-amount	QA	Resolved
[QA-10]	The total-rewards Variable in staking-btc::add-rewards Has a Misleading Name	QA	Resolved
[QA-11]	Use Errors Instead of Panicking	QA	Resolved
[QA-12]	Lack of Event Logging for Sensitive Setters	QA	Resolved
[QA-13]	Use is-in-mainnet to Check if Code is Running on Mainnet	QA	Resolved
[QA-14]	staking-btc::div-down can be simplified	QA	Resolved

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[QA-15]	Reuse get-redeemable-btc-by-amount	QA	Resolved
[QA-16]	Redundant Sender Variable Declaration	QA	Resolved
[QA-17]	Remove Unused Constants	QA	Resolved
[QA-18]	Unused rewards-left Variable	QA	Resolved
[QA-19]	Unused staking-btc::create-order-0-or-fail Function	QA	Resolved
[QA-20]	Redundant Function Wrappers in Stacking-BTC Contract	QA	Resolved

CONTENTS

8. Findings

8.1. Low Findings

[L-01] Do not use `tx-sender` for sensitive operations

Description

Throughout the contract, there are instances where `tx-sender` is used instead of `contract-caller` or passing the caller address.

By doing this, operators who fall victim to phishing scams and interact with malicious contracts can unwittingly interact with the codebase and execute sensitive operations.

For example, if an operator interacts with a malicious contract, that contract can then call the `btc-registry::set-peg-in-sent` function with a maliciously controlled payload.

Another example is a user interacting with a malicious contract, which could then initiate withdrawals on their behalf via the `staking-btc::init-withdraw`.

Of course, the above cases cannot happen without operator and user negligence.

Recommendation

Use `contract-caller` instead of `tx-sender` in all instances outside of the SIP-10 `transfer` function and contract-deployer type variables.

CONTENTS

[L-02] BTCz Token Name, Decimals, and Symbol Should Not Be Changeable

Description

The `BTCz` token contract currently allows for the symbol, decimals, and name of the underlying fungible token to be changed.

The name and symbol, in conjunction with the contract address, should remain immutable. Any external integrator or price aggregator that uses these elements in their UI will cause user confusion if they are ever changed.

Although SIP-10 does not explicitly mention this, it is generally understood that once a fungible token has launched, its name and symbol should never change.

The decimals attribute is critical for any fungible token. Changing it after deployment would have catastrophic implications and must never be done.

Recommendation

Remove the `set-symbol`, `set-decimals`, and `set-name` functions from the `token-btc` contract. Additionally, replace the data variables that hold this information with constants, as they should not be changeable.

CONTENTS

[L-03] Fee Value Can Surpass 100% and Block Operations

Description

The `fee-data` contract contains functionality to change the implicit pegging fees. However, there is no enforcement to ensure that the fee values are valid during the operation of changing the fees. Consequently, the fee values can mistakenly be set to over 100%, causing the `staking-btc` contract operations to revert.

Recommendation

In the `fee-data` contract, when setting the new fees in the `set-peg-out-fee` and `set-peg-in-fee` functions, validate that the new fee does not exceed 100% (i.e., over `ONE_8`).

CONTENTS

[L-04] Reward Commission Can Be Set Over 100% and Block Adding Rewards

Description

The `staking-data` contract includes functionality to change the reward deduction commission. However, there is no validation to ensure that the commission percentage is within a valid range. Consequently, the commission value can mistakenly be set to over 100%, causing the `staking-btc` contract to revert when calling the `add-reward` function.

Recommendation

In the `staking-data` contract, when setting the new commission value in the `set-commission` function, validate that the new fee does not exceed 100% (i.e., over `ONE_8`).

CONTENTS

[L-05] Guard against withdrawal-direct-to-deposit edge-case

Description

A malicious actor can attempt to perform a Denial of Service (DoS) attack on the bridge by initiating a withdrawal with the peg-out address being the same as the peg-in address for deposits.

On-chain, this simply results in user funds being lost, but off-chain, depending on how the bridging software is implemented, it may reach an odd state which can delay further bridge operations.

Recommendation

An on-chain solution would be to validate that the `peg-out-address` argument from the `init-withdraw` function is not an approved peg-in address.

However, due to the extremely unlikely nature of this edge case, adding an on-chain fix would redundantly increase execution costs. As such, the recommendation is to take this situation into consideration in the off-chain bridging software.

CONTENTS

[L-06] Bitcoin Deposits via PUSHDATA2 and PUSHDATA4 Opcodes Are Stuck

Description

As the bridging system is currently implemented, users deposit Bitcoin by transferring BTC to the pegging Bitcoin addresses along with a script output containing the Stacks address that receives the bridged BTCz tokens.

On the Stacks side, any user can call `stacking-btc::deposit` with the transaction buffer from the Bitcoin chain to finalize the deposits and mint BTCz. Both operations are permissionless.

When reading the Stacks deposit address from the Bitcoin transaction on Stacks, the current implementation can handle only directly pushed script data or data pushed using the `PUSHDATA1` opcode.

However, the Stacks address can also be passed in a Bitcoin transaction using the `PUSHDATA2` and `PUSHDATA3` opcodes. If a user attempts to deposit Bitcoin in the bridge using these opcodes, their transaction would be blocked in the Bitcoin address as the `deposit` call on Stacks would revert due to unsupported opcodes.

Recommendation

To support the use of the `PUSHDATA2` and `PUSHDATA4` opcodes, significant alterations must be made to both the `stacking-btc` and `clarity-bitcoin-v1-02` contracts.

It is also generally uneconomical for users to use the `PUSHDATA2` and `PUSHDATA4` opcodes in this case, as it would redundantly increase the fees for the Bitcoin transaction.

As this is a particularly rare case, the recommendation is to clearly document this limitation and implement an off-chain mechanism to allow stuck Bitcoin deposits to be returned to the depositor.

CONTENTS

[L-07] get-redeemable-btc-by-amount and get-redeemable-btc return misleading amount

Description

Using the `get-redeemable-btc-by-amount` and `get-redeemable-btc` functions from the `stacking-btc` contract, a user may estimate how much BTC they would receive after converting their `BTCz`.

However, these functions display the value before fees are d educted, not after. Therefore, the actual amount a user would receive is less.

Recommendation

Document this behavior with an internal comment and create two additional functions with the same logic that return the amounts after fees are deducted.
For example, `get-redeemable-btc-by-amount-after-fees` and `get-redeemable-btc-after-fees` .

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

8.2. QA Findings

[QA-01] Deposits to Contracts with Long Names Are Stuck

Description

As the bridging system is currently implemented, users deposit Bitcoin by transferring BTC to the peg-in Bitcoin addresses along with a script output containing the Stacks principal address that receives the bridged BTCz tokens. The script output uses the `OP_RETURN` plus `PUSHDATA1` opcodes to pass the Stacks principal address.

On the Stacks side, any user can call `stacking-btc::deposit` with the TX buffer from the Bitcoin chain to finalize the deposits and mint BTCz. Both operations are permissionless.

Bitcoin consensus logic currently limits the data returned by `OP_RETURN` to 80 bytes (83 bytes including the 3 overhead bytes). This limit, combined with the Stacks contract address format, enforces a maximum Stacks contract principal name of no more than 52 characters, even though Clarity smart contract names support a maximum length of 128 characters.

This means that depositing BTC using the bridge to contract addresses with names longer than 52 characters is not supported. This limitation is known by the team and will be documented in the bridge documentation.

However, to surpass this limit, some depositors may coordinate with miners to either use the `-datacarriersize` option or Peter Todd's "Libre Relay" patch.

While this is unlikely, if such a deposit is made, depositors need to be aware of another limitation regarding how the contract name is retrieved.

When reading the Stacks deposit address from the Bitcoin transaction on Stacks, although the current implementation extracts the passed address from the `PUSHDATA1` opcode, which supports data up to 255 bytes in length, the actual Clarity smart contract implementation only supports a maximum script size of 128 bytes.

This means that contract addresses with names longer than 102 characters are not supported, even if the deposit bypasses the original 80-byte `OP_RETURN` limit.

To break down why this is so, consider the script data using the default `OP_RETURN` and `PUSHDATA1` opcodes to store a contract principal:

```
6A - OP_RETURN opcode
4C - PUSHDATA1 opcode
L1
    - 1 byte for storing the length of the payload, in this case the Stacks Address, al
< --- from here down, the format belongs to a Stacks contract principal>
03 - A 1-byte type prefix
VV - version of the standard principal that issued the contract
20
    bytes - The 20-byte Hash160 of the standard principal that issued the contract
L2 - A 1-byte length of the contract name, up to 128
<The contract name, as ASCII bytes>
```

1 byte is used for the `OP_RETURN`, another for the `PUSHDATA1` opcode, 1 byte to store the length of the `PUSHDATA1` payload, 2 bytes for the Stacks address type, 20 bytes for its HASH160, and another byte for the length of the contract name. This amounts to 1 + 1 + 1 + 2 + 20 + 1 = 26 bytes.

Thus, the remaining space for the contract name is 128 - 26 = 102 bytes. As Stacks permits the contract name to be 128 characters in length, deposits that bypass the `OP_RETURN` limit with that intent in mind, although unlikely, would remain stuck in the Bitcoin peg-in address.

Recommendation

To support the full use of the `PUSHDATA1` size and a maximal Stacks address, significant alterations must be made to both the `stacking-btc` and `clarity-bitcoin-v1-02` contracts.

Even if the general Bitcoin consensus eventually increases the default `OP_RETURN` limit, making this scenario plausible, it remains a very unlikely edge case for which fixing it on-chain is not justified.

Thus, the recommendation is to document the current bridging system constraints and implement an off-chain mechanism to allow stuck Bitcoin deposits to be returned to the depositor.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-02] Redundant Fee Address Logic on Stacks

Description

In the `fee-data` contract, a principal `fee-address` variable is managed. Additionally, there is a wrapper getter in the `staking-btc` for this value.

As the bridging system is implemented, fees are initially kept on the Bitcoin side in the pegging-in address. The fees are retained at the sources.

A fee address is irrelevant in the context of the Stacks blockchain, making it redundant code.

Recommendation

Remove the `fee-address` logic from the entire codebase.

CONTENTS

[QA-03] Consider Moving Commission Logic Off-Chain

Description

The `staking-btc` contract provides functionality to add extra BTC to the internally tracked `total-btc` amount via the `add-rewards` function.

In practice, this function is called after the team has added or donated extra BTC tokens to the peg-out Bitcoin address, thereby increasing the overall value of the BTCz token.

A particular aspect of the `add-rewards` logic is that it deducts a commission amount from the input BTC amount and only increments the total underlying BTC accounting value with the remaining amount after the commission is deducted.

Recommendation

Since there is no other on-chain logic tied to the commission logic, it may be simplest to move it completely off-chain. As only the team can add rewards, they can choose the amount to add. They can already keep a portion as a commission and only call the `add-rewards` function with the amount after the commission is taken.

As it stands, the on-chain commission logic is used merely as a display for users to determine how much commission the team has kept out of a reward system that the team controls. Thus, it does not make sense for it to exist on- chain.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-04] Pausing Pegging In/Out Functionality Should Not Be in the Fee-Data Contract

Description

The `fee-data` contract contains logic related to fees, such as setting and getting the pegging in and pegging out fees, the peg out gas fee, and the fee address recipient.

However, it also includes logic for pausing and unpausing pegging in/out. This functionality is not fee-related and, as such, is out of place in the `fee-data` contract.

Recommendation

One option is to move the pausing pegging in/out functionality to a different contract. By doing so, you also gain the granularity of having a separate principal responsible for pausing/unpausing, different from the one responsible for changing the fees.

Another option is to change the `fee-data` contract name to a more inclusive one, such as `peg-manager`.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-05] Revoking Withdrawals Not Implemented

Description

The current withdrawal data structure, which is used to save any executed withdrawal on-chain, includes an unused `revoked` field.

Recommendation

If the `revoked` field is intended for use in a future version, please acknowledge this issue. Otherwise, implement the missing functionality.

CONTENTS

[QA-06] Overlapping Error Code Ranges

Description

Within the codebase, each contract has a unique error code range to easily identify the contract from which the error was sent.

`btc-registry`	`2000-2999`
`clarity-bitcoin-v1-02`	`1000-1999`
`fee-data`	`3000-3999`
`ft-trait`	`None`
`stacking-btc`	`1000-1999`
`stacking-data`	`4000-4999`
`token-btc`	`5000-5999`

However, the `staking-btc` contract shares the same range as the `clarity-bitcoin-v1-02` . Since `staking-btc` calls functions from the `clarity-bitcoin-v1-02` contract, this overlap can cause confusion when debugging failed transactions.

Recommendation

Change the `staking-btc` contract error code range to one that is not already used, such as 6000-6999. Keep in mind that ranges below 1000 are typically reserved for SIP standards and internal Stacks functions.

CONTENTS

[QA-07] Incorrectly Referencing BTCz as sBTC in Code

Description

Within the `staking-btc` codebase, there are references to the `BTCz` token (contract token-btc) as both `btcz` and `sbtc`. This creates slight confusion for any integrator, as the `sBTC` Stacks token may be misinterpreted when reading the codebase.

Recommendation

Change all occurrences of `sbtc` to `btcz` within the `staking-btc` contract.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-08] Add Fee to `stacking-btc::deposit` Print Call for Better Off-Chain Tracking

Description

When a deposit is made on the Bitcoin chain and the `stacking-btc::deposit` function is called on the Stacks blockchain, a fee in BTC is retained in the Bitcoin address. Only the remaining equivalent BTC is converted to BTCz.

Recommendation

To help the team more easily identify this amount, which they can retain on the Bitcoin side, add the `fee` variable amount to the `print` statement in the `stacking-btc::deposit` function.

CONTENTS

[QA-09] Misleading Amount Argument Name in stacking-btc::get-redeemable-btc-by-amount

Description

The `stacking-btc::get-redeemable-btc-by-amount` function accepts an amount of BTCz tokens, which are then converted to Bitcoin (BTC) tokens using the internal BTC to BTCz ratio.

However, the function's amount variable is mistakenly named `btc-amount` instead of `btcz-amount`. This may lead to confusion for any third-party integrators.

Recommendation

Rename the `btc-amount` argument in `stacking-btc::get-redeemable-btc-by-amount` to btcz-amount.

CONTENTS

[QA-10] The `total-rewards` Variable in `staking-btc::add-rewards` Has a Misleading Name

Description

In the `staking-btc::add-rewards` function, the `total-rewards` variable stores the new, up-to-date total BTC amount, which includes the newly added reward amount.

```
(total-rewards (+ (get-total-btc) rewards))
```

The name is misleading as it does not represent the total reward amount that was added.

The variable is also emitted from a `print` command with a misleading label.

```
(
print{action:"add-rewards",
data:{final-commission:final-commission,
total-rewards:total-rewards}}
)
```

Recommendation

Rename the `total-rewards` variable to a more appropriate name, such as `new-total-btc` . Additionally, consider implementing separate logic for tracking the total deposited rewards if that is required.

CONTENTS

[QA-11] Use Errors Instead of Panicking

Description

In the `staking-btc` contract, there are instances where `unwrap-panic` is used instead of `unwrap!` with a custom error.

Specifically, it is logically meaningful to add custom errors in the following four instances:

- In the `deposit` function, when accessing an index outside of the transaction output array bounds.
- In `decode-order-0-or-fail`, in both instances when slicing an invalid order-script buffer.

Ending execution in a panic results in a runtime error. Runtime errors cannot be handled by the caller and do not provide any meaningful information about the execution. Therefore, they are discouraged.

Recommendation

Use `unwrap!` with a custom error instead of `unwrap-panic` in the mentioned instances.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-12] Lack of Event Logging for Sensitive Setters

Description

Throughout the codebase, several sensitive setters do not log information when called, which hinders off-chain parsers from tracking changes.

Adding logs that capture the old and new values would assist off-chain trackers in creating a comprehensive timeline of the bridge's state and activity.

Recommendation

Implement logging with relevant information in all sensitive setters within the codebase.

CONTENTS

[QA-13] Use `is-in-mainnet` to Check if Code is Running on Mainnet

Description

When verifying that the current code is running on the Stacks mainnet, use the internal keyword `is-in-mainnet` instead of comparing the chain ID to 1 with `(is-eq chain-id u1)`.

Using `is-in-mainnet` enhances code readability and reduces execution costs.

Recommendation

Use `is-in-mainnet` to check if the code is running on the mainnet.

CONTENTS

[QA-14] staking-btc::div-down can be simplified

Description

The `staking-btc::div-down` function has two arguments, `a` and `b`, and:

1. Checks if `a` is 0 and, if so, returns 0.
2. Otherwise, returns `(a * 1e8 / b)` .

```
(define-read-only (div-down (a uint) (b uint))
        (if (is-eq a u0)
                u0
                (/ (* a ONE_8) b)))
```

The first operation, checking if `a` is 0, is redundant for several reasons:

- If `a` is 0, the result of the second branch would also be 0.
- It actually increases the execution fee over time, which will be elaborated on.

`div-down` is called in two places: directly from the `deposit` function and in the `get-btc-to-sbtc-ratio` function.

In the `deposit` function, `div-down` is called with `a` as the BTC amount after fees. This is never 0 because, to be a successful transfer on the Bitcoin network, a minimum amount greater than 0 will always exist due to the implicit dust limit, and the subtracted Zest fee is a percentage of that.

Within the `get-btc-to-sbtc-ratio` function, the only time `div-down` is called with 0 for `a` is when there has not been even one BTC deposit into the bridge. In that case, and only that case, it acts as a slight fee optimization.

However, after one BTC bridge deposit, it will always be called redundantly, increasing the fees on each call.

Recommendation

Remove the `if` statement from the `div-down` function and keep only the `else` branch logic.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-15] Reuse get-redeemable-btc-by-amount

Description

In the `staking-btc::init-withdraw` function, the redeemable amount is retrieved as follows:

```
(btc-to-sbtc-ratio (get-btc-to-sbtc-ratio))
(redeemable-btc (mul-down btcz-amount btc-to-sbtc-ratio))
```

However, there is already a function `staking-btc::get-redeemable-amount` that performs the same operation, resulting in duplicated code within the `init-withdraw` function.

Recommendation

Reuse the `get-redeemable-btc-by-amount` function when executing the `init-withdraw` operation.

CONTENTS

[QA-16] Redundant Sender Variable Declaration

Description

In the `staking-btc::deposit` function, the sender is declared within a `let` command as `(sender recipient)` and subsequently used when minting the BTCz tokens. Adding a new variable that simply copies an existing principal is redundant.

Additionally, from a contextual standpoint, using `recipient` is more appropriate since it represents the entity that will receive the BTCz tokens on the Stacks blockchain, rather than the entity that initiated the deposit on the Bitcoin block.

Recommendation

Remove the `(sender recipient)` declaration and use `recipient` instead of `sender`.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-17] Remove Unused Constants

Description

In the `staking-btc` contract, there are two unused constants: `err-address-mismatch` and `err-tx-mined-before-request`.

Recommendation

Remove the unused constants. Then, adjust the remaining constant values to be incremental (i.e., from *u1000* to *u1008* without gaps).

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-18] Unused `rewards-left` Variable

Description

In the `staking-btc::add-rewards` function, the `rewards-left` variable is declared using a `let` command but is never utilized.

Recommendation

Remove the unused `rewards-left` variable.

CONTENTS

[QA-19] Unused staking-btc::create-order-0-or-fail Function

Description

Within the `staking-btc` contract, the `create-order-0-or-fail` function is unused and does not serve any useful off-chain calculation, making it redundant.

Recommendation

Remove the `staking-btc::create-order-0-or-fail` function.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
sBTC

[QA-20] Redundant Function Wrappers in Stacking-BTC Contract

Description

In the `stacking-btc` contract, there are multiple function wrappers that merely pass through calls to the `fee-data`, `btc-registry`, or `staking-data` contracts.

Examples:

```
(define-read-only (is-peg-in-paused)
        (contract-call? .fee-data is-peg-in-paused))

(define-read-only (is-peg-in-address-approved (address (buff 128)))
        (contract-call? .btc-registry is-peg-in-address-approved address))

(define-read-only (get-withdrawal-or-fail (id uint))
        (contract-call? .stacking-data get-withdrawal-or-fail id))
```

With the exception of two function wrappers, `get-total-btc` and `set-total-btc`, all other existing function wrappers are called only once from the `stacking-btc` contract. This redundancy increases the contract code size without providing any benefit. Each of the underlying functions can be called directly without the need for these wrappers.

Recommendation

Remove the following functions from the `stacking-btc` contract and inline their original calls in the code:

- `is-peg-in-paused`
- `is-peg-out-paused`
- `get-peg-in-fee`
- `get-peg-out-fee`
- `get-peg-out-gas-fee`
- `get-fee-address` (this function isn't even called at all)
- `is-peg-in-address-approved`
- `get-peg-in-sent-or-default`
- `get-withdrawal-or-fail`
- `get-commission`
- `get-commission-total`
- `get-withdrawal-nonce`
- `set-commission-total`
- `set-withdrawal-nonce`
- `set-withdrawal`