ClarityAlliance

sBTC SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA), MARCHEV

JANUARY 5TH, 2025

CONTENTS

ClarityAlliance
Security Review

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

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
sBTC

3. Introduction

A time-boxed security review of sBTC, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About sBTC

sBTC is a 1:1 Bitcoin-backed asset that enables users to utilize their BTC within DeFi, decentralized applications, and other blockchain-based ecosystems. Designed to bring Bitcoin's security and liquidity into programmable environments, sBTC expands Bitcoin's utility beyond simple transactions.

Bitcoin Finality
All Stacks transactions, including those that involve sBTC, benefit from 100% Bitcoin finality. This means transactions on Stacks, once confirmed, are as irreversible as Bitcoin's.

Programmability
Access a world of new use cases for Bitcoin thanks to Clarity, a full featured smart contract language optimized for security and predictability.

Censorship-resistance
sBTC operations happen on the Bitcoin main chain, meaning that external actors cannot censor these operations.

Source: Stacks sBTC

CONTENTS

ClarityAlliance
Security Review
sBTC

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.

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

6. Security Assessment Summary

This audit focused on a subset of the sBTC system, specifically components related to Clarity and Emily, rather than the full codebase. The primary emphasis was on deposit functionality, as the system was not yet live for withdrawals at the time of review.

- Initial analysis started at 53cc756c0ddecff7518534a69bef59fadb5ab1d4
- Subsequent analysis was conducted from 5c850909440a2563e8d1450696b853747ef55bf0

ClarityAlliance
Security Review

sBTC

CONTENTS

ClarityAlliance
Security Review
sBTC

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA), Marchev engaged with - to review sBTC. In this period of time a total of **27** issues were uncovered.

Protocol Summary

Protocol Name	sBTC
Date	January 5th, 2025

Findings Count

Severity	Amount
High	1
Medium	4
Low	5
QA	17
Total Findings	**27**

CONTENTS

ClarityAlliance
Security Review

sBTC

Summary of Findings

ID	Title	Severity	Status
[H-01]	Random Deposits Can Be Added Through Inadequately Validated Deposit Creation API	High	Partially Resolved
[M-01]	sBTC Balance Logic Causes External Integration Issues	Medium	Acknowledged
[M-02]	Inability to Rotate Signers to Standard Principals with More Than 15 Keys	Medium	Resolved
[M-03]	sBTC Withdrawal Mechanism Is Susceptible to Griefing Attacks	Medium	Acknowledged
[M-04]	Random Withdrawals Can Be Added Through Inadequately Validated Withdrawal Creation API	Medium	Acknowledged
[L-01]	Duplicate Entry in BUFF_TO_BYTE	Low	Resolved
[L-02]	Inconsistent Handling of Dust Limit on Deposits and Withdrawals	Low	Acknowledged
[L-03]	Authorization Mechanism Is Poorly Applied in The sBTC Contract	Low	Acknowledged
[L-04]	Retry Mechanism Lacks Exponential Backoff Strategy	Low	Acknowledged
[L-05]	Resolve Outstanding Critical TODOs and Missing Features	Low	Acknowledged

CONTENTS

ClarityAlliance
Security Review
sBTC

Summary of Findings

ID	Title	Severity	Status
[QA-01]	Incomplete sBTC Logging on Deposit Creation	QA	Acknowledged
[QA-02]	Typographical Errors	QA	Resolved
[QA-03]	Remove Unused Constants	QA	Resolved
[QA-04]	Token Symbol Variable Length May Constrain Future Symbol Updates	QA	Acknowledged
[QA-05]	sBTC Token Name and Symbol Should Not Be Changeable	QA	Acknowledged
[QA-06]	Simplification Opportunities in sBTC Operations	QA	Acknowledged
[QA-07]	sBTC Contracts Structure and Style Inconsistencies	QA	Acknowledged
[QA-08]	Use Constants Where Appropriate	QA	Acknowledged
[QA-09]	Misleading, Outdated, or Incomplete Comments in sBTC Contracts	QA	Acknowledged
[QA-10]	sBTC Protocol Contract Type Can Be Updated With Arbitrary Data	QA	Acknowledged
[QA-11]	Redundant Protocol Mapping in sBTC Contracts	QA	Acknowledged
[QA-12]	Some sBTC Protocol Setter Functions Lack Corresponding Role	QA	Acknowledged
[QA-13]	Outdated Emily API Domain Name Generation Schema Documentation for Non-Production Environments	QA	Resolved
[QA-14]	Simplification of EmilyStackUtils Operations	QA	Acknowledged
[QA-15]	Cleanup EmilyStack Class	QA	Acknowledged
[QA-16]	Typographical Errors in Emily	QA	Resolved
[QA-17]	Use Constants Instead of Magic Numbers in Emily APIs	QA	Acknowledged

CONTENTS

ClarityAlliance
Security Review

sBTC

8. Findings

8.1. High Findings

[H-01] Random Deposits Can Be Added Through Inadequately Validated Deposit Creation API

Description

The Emily API routes, which handle the creation of orders such as deposits or withdrawals, ultimately write data into the underlying DynamoDB database. Currently, creating a deposit through the API is not restricted by any permission system. Without such restrictions, an attacker can intentionally spam the creation of deposit HTTP requests, leading to database saturation, increased AWS costs, and potential database issues.

When creating a deposit via the `POST /deposit` → `handlers::deposit::create_deposit`:

- There is no requirement for an authorization mechanism or API key.
- The `POST` body is a JSON object mapped to a `CreateDepositRequestBody` type entry.

```
pub struct CreateDepositRequestBody {
    /// Bitcoin transaction id.
    pub bitcoin_txid: String,
    /// Output index on the bitcoin transaction associated with this specific
    // deposit.
    pub bitcoin_tx_output_index: u32,
    /// Reclaim script.
    pub reclaim_script: String,
    /// Deposit script.
    pub deposit_script: String
}
```

- Several validations are missing in the payload body:
  - There is no validation on the `bitcoin_txid` to ensure it is a valid Bitcoin transaction. Any arbitrary string can be passed.
  - There is no validation on the `bitcoin_tx_output_index` to confirm it corresponds to an index that exists in the `bitcoin_txid`.

- While the `reclaim_script` and `deposit_script` must be validly `sbtc::deposits::DepositScriptInputs` formatted, there is no enforced correspondence between the Bitcoin transaction and these scripts. As `reclaim_script` and `deposit_script` are Non-Key Attributes, there is no constraint preventing the reuse of the script.
- Data is only validated to ensure it was correctly saved, which is always true for newly created deposits.
- Finally, the data is directly saved in the database without any further checks.

CONTENTS

ClarityAlliance
Security Review

sBTC

An attacker can easily spam the `/deposit` endpoint with arbitrary inputs, which do not even need to be valid Bitcoin transaction IDs, as long as the payload includes a valid reclaim and deposit script. The same script can be reused for each randomly generated `POST`.

This attack not only increases database costs but can also lead to a denial of service (DoS) under high load.

Recommendation

Implement permissioning to allow only trusted API key holders to add deposits and introduce further validations on the Bitcoin transaction itself. Ensure checks are in place to:

- Verify that `bitcoin_txid` is a valid Bitcoin transaction string.
- Confirm that `bitcoin_tx_output_index` is an output index that exists in the specific transaction.
- Ensure `reclaim_script` and `deposit_script` are actually connected to the `bitcoin_txid`.

This issue was also identified by the `sBTC` team and is a work in progress.

ClarityAlliance
Security Review

sBTC

8.2. Medium Findings

[M-01] sBTC Balance Logic Causes External Integration Issues

Description

Users holding sBTC can withdraw it from Stacks to Bitcoin by calling the `sbtc-withdrawal::initiate-withdrawal-request` function. The withdrawal process involves several steps:

1. The user initiates the withdrawal, specifying the amount they wish to withdraw and the maximum fee they are willing to pay (in BTC).
2. The native Stacks core logic attempts to finalize the transaction on Bitcoin or refunds it on Stacks if it fails or cannot be executed.

After step (1) is completed, the user is recorded in sBTC's internal accounting as having locked tokens.

```
(try! (ft-burn? sbtc-token amount owner))
(ft-mint? sbtc-token-locked amount owner)
```

However, the user's balance, as perceived by external integrators when calling the standard SIP-10::get-balance function, does not change:

```
(define-read-only (get-balance (who principal))
      (ok (+ (ft-get-balance sbtc-token who) (ft-get-balance sbtc-token-locked who))
)
```

Due to this mechanism, between the initiation of a withdrawal and its finalization or execution, users' principals appear to have a certain balance externally. However, attempting to transfer this balance will fail, as the `transfer` function only recognizes unlocked sBTC tokens as available for transfer.

Consider scenarios where:

- A user initiates a withdrawal and then
- Interacts with a third-party protocol that accepts generic `SIP-10` tokens, including `sBTC`, for example, for depositing.
- The protocol deposit would fail at the `transfer` call due to insufficient tokens, even if `get-balance` internally indicates the user has enough tokens. This results in poor UI/UX and can be difficult to debug initially.

While there is no token loss, having `get-balance` implemented in this way requires protocols wishing to integrate with it to call `sbtc-token::get-balance-available` instead. This significantly increases integration friction, as it deviates from standard behavior.

There is no benefit for any third-party protocol to know how much sBTC a user currently has locked.

Another point to consider is that users can create always-reverting withdrawals by setting the maximum fee to 0. Withdrawals without a fee will generally be rejected. Although uncommon, there may be potential situations where this can be exploited.

CONTENTS

ClarityAlliance
Security Review

sBTC

Recommendation

Modify `get-balance` to return the balance of the underlying `sbtc-token`. Remove the `get-balance-available` function and add a `get-balance-locked` function to show how much users have locked at that moment.

CONTENTS

[M-02] Inability to Rotate Signers to Standard Principals with More Than 15 Keys

Description

When a new signer principal and subsequent data need to be changed, the `sbtc-bootstrap-signers::rotate-keys-wrapper` function is invoked. This function allows the current signer principal to be changed to a standard principal, which can consist of up to 128 public keys.

However, the implementation fails when more than 15 keys are provided due to an incorrect maximum iterator buffer length in the `(concat-pubkeys-fold` function.

```
;; Concatenate a pubkey buffer with a length prefix.
;; The max size of the iterator is 4239 bytes, which is (33 * 128) 4224 bytes
;; for the public keys and 15 bytes for the length prefixes.
(define-read-only (concat-pubkeys-fold (pubkey (buff 33)) (iterator (buff 510)))
  (let
    (
      (pubkey-with-len (concat (bytes-len pubkey) pubkey))
      (next (concat iterator pubkey-with-len))
    )
    (unwrap-panic (as-max-len? next u510))
  )
)
```

The maximum iterator size can reach 4352 bytes, calculated as: 33 (pubkey length) * 128 (maximum number of keys) + 128 (one byte length prefix for each address). However, the code currently uses the value 510, which restricts the algorithm to 510/(33 key size + 1 byte length prefix) ⇒ 15 keys.

This limitation significantly impacts the functionality of the API. If, in the future, the community requires more than 15 signers, it will not be possible.

Recommendation

Adjust the iterator buffer length from 510 to 4352, and update the `as-max-len?` max_length parameter to `u4352`.

Note: The comments within the function are also incorrect, but these are addressed in another issue.

Issue was also identified by the `sBTC` developers and resolved before deployment.

CONTENTS

ClarityAlliance
Security Review

sBTC

[M-03] sBTC Withdrawal Mechanism Is Susceptible to Griefing Attacks

Description

Users holding sBTC can withdraw it from Stacks to Bitcoin by calling the `sbtc-withdrawal::initiate-withdrawal-request` function. The withdrawal process involves several steps:

- The user initiates the withdrawal, specifying the amount they wish to withdraw and the maximum fee they are willing to pay (in BTC).
- The native Stacks core logic attempts to finalize the transaction on Bitcoin or refund it on Stacks if it fails or cannot be executed.

For step (2), the current signer principal must either call `accept-withdrawal-request` when the withdrawal is correct and executable or call `reject-withdrawal-request` to reject the withdrawal. Accepting and rejecting are necessary to burn or unlock the user's pending `sBTC`.

It is important to note that when users initiate a withdrawal, they must specify the maximum fee they are willing to pay for the transaction to be executed on the Bitcoin network.

This mechanism is susceptible to griefing because a malicious attacker can continuously initiate withdrawals with a maximum fee set to 0. These requests will ultimately be rejected, as without a fee, the underlying peg-out mechanism will eventually lose funds. However, by rejecting the request, i.e., calling `reject-withdrawal-request`, the principal signer incurs an execution fee on Stacks.

The overall execution fee for initiating a withdrawal is comparable to that of rejecting one. Even if bulk rejection is used via the `complete-withdrawal` function, the attacker does not face a significant cost-to-damage ratio concerning block costs.

Action	Write Length	Write Count	Read Length	Read Count	Runtime
`initiate-withdrawal-request`	216	6	38,425	22	78,005
`reject-withdrawal-request`	22	5	59,301	31	104,363
`complete-withdrawal` (average on 300 rejected requests)	22	5	45,897	28	11,607

In practice, the effectiveness of the attack, meaning the attacker's loss compared to the signer's loss (or cost-to-damage ratio), depends on the aforementioned block costs coupled with dynamic execution fees, which are network-determined.

Regardless of the cost to the attacker, allowing this situation to persist will result in rejected transactions, even if caused by the mistake of organic users, which only the protocol signer can execute.

CONTENTS

ClarityAlliance
Security Review

sBTC

Recommendation

Implement a minimum max fee value for initiating withdrawal requests. Initially, it can be set to 0 and only increased if the indicated attack is observed in the wild (ITW).

CONTENTS

ClarityAlliance
Security Review

sBTC

[M-04] Random Withdrawals Can Be Added Through Inadequately Validated Withdrawal Creation API

Description

The Emily API includes routes for creating orders, such as deposits or withdrawals, which ultimately write data to the underlying DynamoDB database.

Currently, creating a withdrawal through the API is not restricted by an API key or any permission system. Without such restrictions, an attacker can intentionally spam the API with HTTP requests, leading to database saturation, increased AWS costs, and potential loss of funds.

When a withdrawal entry is created via the `POST /withdrawal` → `handlers::withdrawal::create_withdrawal` :

- There is no requirement for an authorization mechanism or API key.
- The `POST` body is a JSON object mapped to a `CreateWithdrawalRequestBody` type entry.
- All data to be saved in the database is directly taken from the request body.
- The data is validated only to ensure no corruption with previously saved data, a check that always passes for new entries.
- Finally, the data is directly saved into the database withdrawal table.

An attacker can exploit the `/withdrawal` endpoint with arbitrary inputs due to the absence of any checks, leading to increased database costs and potential denial of service (DoS) attacks on the database.

Recommendation

Implement a permission system to ensure that only holders of trusted API keys can add withdrawals. Additionally, enhance the validation of the payload itself. Specifically, add checks to the `CreateWithdrawalRequestBody` payload:

- `stacks_block_hash` must be a valid Stacks block hash.
- `stacks_block_height` must correspond to the indicated `stacks_block_hash` .
- `recipient` must be a valid Bitcoin address.

ClarityAlliance
Security Review
sBTC

8.3. Low Findings

[L-01] Duplicate Entry in BUFF_TO_BYTE

Description

When a new signer principal is added and subsequent data changes are required, the `sbtc-bootstrap-signers::rotate-keys-wrapper` function is invoked. Within this function, `pubkeys-to-principal` is called to generate the corresponding Stacks standard principal derived from the provided public keys.

During the execution of `pubkeys-to-principal`, the `uint-to-byte` function is used twice to convert unsigned integers to bytes. This function relies on a byte array, `BUFF_TO_BYTE`, which is incorrectly implemented. Specifically, `BUFF_TO_BYTE` returns the same value (`0x33`) for both the 50th and 51st elements, meaning that converting the integer 50 results in the byte value 51.

Although the function is incorrectly implemented, there is no direct impact at present, as there is no call that uses 50 as an input value.

In one instance, within the `pubkeys-to-spend-script` function, it is always called with an 80 offset, ensuring that the value 50 is never reached:

```
(concat (uint-to-byte (+ u80 m)) ;; "m" in m-of-n
  (concat (uint-to-byte (+ u80 (len pubkeys))) ;; "n" in m-of-n
```

In another instance, the `bytes-len` function is called with a buffer length of at most 33, again avoiding the value 50.

```
(define-read-only (bytes-len (bytes (buff 33)))
  (unwrap-panic (element-at BUFF_TO_BYTE (len bytes)))
)
```

Recommendation

In the `BUFF_TO_BYTE` array, insert the correct value `0x32` at the 50th index position.

CONTENTS

[L-02] Inconsistent Handling of Dust Limit on Deposits and Withdrawals

Description

To ensure the Bitcoin network accepts a BTC transfer, the minimum number of satoshis that can be transferred, known as the dust limit, is verified during each withdrawal and deposit. However, there are inconsistencies in how this limit is interpreted and in the accompanying comments.

For `deposits` :

- No comment is provided at the declaration.
- A comment at the check location indicates the amount must be strictly
- greater than the dust limit.
- Despite this, the dust limit is allowed to be met.

```
(define-constant dust-limit u546)

;; Check that amount is greater than dust limit
(asserts! (>= amount dust-limit) ERR_LOWER_THAN_DUST)
```

For `withdrawals` :

- A comment at the declaration suggests the limit should be allowed for withdrawal.
- No comment is present at the check location.
- The dust limit is not allowed to be met.

```
;; The minimum amount of sBTC you can withdraw
(define-constant DUST_LIMIT u546)

(asserts! (> amount DUST_LIMIT) ERR_DUST_LIMIT)
```

Due to the current implementation, users can deposit amounts equal to the dust limit but cannot withdraw them. Consequently, users who deposit dust-limit amounts will need to deposit additional satoshis to withdraw.

Recommendation

Ensure the dust limit is interpreted consistently in both scenarios. Additionally, update the comments to reflect this uniform interpretation.

CONTENTS

[L-03] Authorization Mechanism Is Poorly Applied in The sBTC Contract

Description

The new authorization schema of the `sBTC` is implemented using an `active-protocol-contracts` map to associate protocol roles with contract principals, and an `active-protocol-roles` map to associate, in reverse, the contract principal with the active role.

This new implementation is correctly utilized in the `sbtc-registry` contract to ensure that each specific contract can only perform its intended operations.

However, in the `sbtc-token` contract, all `protocol-*` functions are checked in such a way that any authorized contract type can perform any action as long as it is recognized as itself.

For instance, consider the `sbtc-token::protocol-lock` function, which is invoked when locking sBTC before a withdrawal:

```
(define-public (protocol-lock (amount uint) (owner principal) (contract-flag
  (buff 1)))
  (begin
    (try!
      (contract-call? .sbtc-registry is-protocol-caller contract-flag contract-caller)
    (try! (ft-burn? sbtc-token amount owner))
    (ft-mint? sbtc-token-locked amount owner)
  )
)
```

The `sbtc-registry::is-protocol-caller` function is called with the data provided by the contract. This implies that any future deposit contract can implement and call this function, as long as it passes its role (depositor).

This completely undermines the purpose of such a mechanism, as some functions should not be arbitrarily called by any approved contract.

Recommendation

Implement specific role-checking functions in `sbtc-registry`, such as `is-protocol-withdrawer` and `is-protocol-depositor`, and apply them accordingly to each `protocol-*` function in the `sbtc-token` contract.

CONTENTS

[L-04] Retry Mechanism Lacks Exponential Backoff Strategy

Description

The Emily API has implemented a retry mechanism for handling database update failures in several scenarios:

- When adding a chainstate (`POST /chainstate`) , the execution flow reaches `accessors::add_chainstate_entry_with_retry` .
- When updating deposits (`PUT /deposit`), the flow reaches `accessors::pull_and_update_deposit_with_retry` .
- When updating withdrawals (`PUT /withdrawal`) , it reaches `accessors::pull_and_update_withdrawal_with_retry` .

In each of these instances, the retry mechanism does not incorporate a backoff period between retries, resulting in immediate retry attempts within a loop:

```
for _ in 0..retries
```

This approach can lead to high CPU usage and potential throttling of the database service, and in extreme cases, it may even cause database failures.

Recommendation

Implement an exponential backoff retry mechanism with jitter.

CONTENTS

[L-05] Resolve Outstanding Critical TODOs and Missing Features

Description

Throughout the Emily codebase, there are 34 different TODOs. These TODOs highlight a range of issues, from missing unit tests:

```
/// TODO(393): Add handler unit tests.
```

to critical issues that need resolution before deployment:

```
/// TODO
//(TBD): This is the only value that will work at the moment because the API needs to
/// TODO(TBD): Get the amount from some script related data somehow.
```

There is essential functionality that currently does not work, such as obtaining the deposit API amount, which is noted with a TODO and must be addressed before deployment. See the relevant code here.

Recommendation

Resolve the TODOs before deployment. If resolving all is not feasible, at least address the known critical issues, as the project cannot be deployed otherwise.

CONTENTS

8.4. QA Findings

[QA-01] Incomplete sBTC Logging on Deposit Creation

Description

In the `sbtc-registry` contract, a `print` command is included in each public function to ensure proper event emissions. In nearly all cases, all arguments are printed. However, the `complete-deposit` function is an exception, as the `recipient` principal is not emitted.

The absence of this parameter increases the difficulty for off-chain systems to monitor `sBTC` transactions.

Recommendation

Include the `recipient` in the `print` command within the `sbtc-registry::complete-deposit` function.

CONTENTS

[QA-02] Typographical Errors

Description

There are several typographical errors and opportunities for slight wording improvements throughout the codebase within scope:

- `sbtc-registry#L25` :The word `withdrawaled` should be corrected to `withdrawn` .
- `holdings.clar#L119` : Although `unexistent` is a valid term, it is less commonly used. Replacing it with `nonexistent` would be more appropriate.

Recommendation

Make all the suggested changes.

ClarityAlliance
Security Review
sBTC

CONTENTS

[QA-03] Remove Unused Constants

Description

Within the codebases under review, there are several instances of unused constants:

- In `sbtc-registry` : `ERR_INVALID_REQUEST_ID` and `ERR_MULTI_SIG_REPLAY`
- In `sbtc-token` : `ERR_NOT_AUTH`
- In `sbtc-withdrawal` : `MAX_ADDRESS_VERSION_BUFF_32`

Recommendation

Remove these constants to enhance code readability, reduce clutter, and slightly decrease runtime read counts and costs. Additionally, after their removal, rebase the error codes to eliminate any gaps.

ClarityAlliance
Security Review
sBTC

CONTENTS

[QA-04] Token Symbol Variable Length May Constrain Future Symbol Updates

Description

The `sbtc-token` contract currently defines `token-symbol` as `(string-ascii 10)` and includes functionality to update it using `protocol-set-symbol`.

However, SIP-10 specifies that token symbols should be of type `(string-ascii 32)`. Although the current symbol fits within the 10-character limit, this restriction could hinder future updates to longer symbols that would be valid under SIP-10.

Recommendation

Modify the `token-symbol` variable type to `(string-ascii 32)` to align with SIP-10 specifications and provide maximum flexibility for future symbol updates.

ClarityAlliance
Security Review
sBTC

CONTENTS

[QA-05] sBTC Token Name and Symbol Should Not Be Changeable

Description

The `sBTC` token contract currently allows for the modification of the token's name and symbol.

The name and symbol, along with the contract address, should remain immutable. Changing these elements can lead to user confusion for any external integrators or price aggregators that rely on them in their user interfaces.

While SIP-10 does not explicitly address this, it is generally understood that once a fungible token is launched, its name and symbol should remain unchanged.

Recommendation

Remove the `protocol-set-name` and `protocol-set-symbol` functions from the `sbtc-token` contract. Additionally, convert the variables storing this data into constants to ensure they remain unchangeable.

CONTENTS

ClarityAlliance
Security Review
sBTC

[QA-06] Simplification Opportunities in sBTC Operations

Description

Within the `sBTC` contracts, there are several opportunities for implification that can enhance code readability, reduce code size, and potentially lower runtime execution costs.

In `sbtc-withdrawal`:

- `L177`: The check `(is-eq (- requested-max-fee fee) u0)` can be simplified to a direct comparison `(is-eq requested-max-fee fee)`.
- `L254`: The variable `current-request-id` is unused, and is `(get request-id withdrawal)` redundantly called again.
- Use `sbtc-registry::get-current-signer-principal` instead of `get-current-signer-data` in the functions `accept-withdrawal-request`, `reject-withdrawal-request`, and `complete-withdrawals`.

In `sbtc-deposit`:

- Use `sbtc-registry::get-current-signer-principal` instead of `get-current-signer-data` in `complete-deposit-wrapper`.

The entire `ERR_<ACTION>_INDEX_PREFIX` error system can be simplified wherever it appears:

- In all instances, the error code calculation, such as for withdrawals: `(err (+ ERR_WITHDRAWAL_INDEX_PREFIX (+ u10 index)))`, can be rewritten using a single `+` operation: `(err (+ ERR_WITHDRAWAL_INDEX_ index))`.
- The constants `ERR_KEY_SIZE_PREFIX`, `ERR_DEPOSIT_INDEX_PREFIX`, and `ERR_WITHDRAWAL_INDEX_PREFIX` are declared with the direct attribution of an error value immediately next to them.

Example for `sbtc-bootstrap-signers`:

```
(define-constant ERR_KEY_SIZE_PREFIX (unwrap-err! ERR_KEY_SIZE (err true)))
(define-constant ERR_KEY_SIZE (err u200))
```

In each case, the paired error code is never used, and when using the `ERR_<ACTION>_INDEX_PREFIX` value, an extra offset of 10 is added. Instead of the elaborate schema, directly hardcode the calculated value with the added index, avoiding an extra addition during a failed bulk iteration call.

```
(define-constant ERR_KEY_SIZE_PREFIX u210)
;; ... code ...
(err (+ ERR_KEY_SIZE_PREFIX index))
```

Recommendation

Implement the suggested changes.

CONTENTS

ClarityAlliance
Security Review
sBTC

[QA-07] sBTC Contracts Structure and Style Inconsistencies

Description

The Clarity contracts that make up `sBTC` generally follow a consistent structure and coding style, except for `sbtc-token`, which is intentionally written in a simpler manner. However, there are minor differences and inconsistencies among the contracts:

1. `sbtc-bootstrap-signers` and `sbtc-deposit` use lowercase constants, while `sbtc-withdrawal` uses uppercase constants.
2. The contracts slightly deviate from a common contract layout. Specifically:
   - `sbtc-bootstrap-signers` uses `errors` as a header, whereas `sbtc-deposit` uses `error codes`, and `sbtc-registry` uses `Error codes`.
   - Headers in `sbtc-registry` start with uppercase letters and differ by using `Maps` (as opposed to `data maps`) and `Variables` (as opposed to `vars`).
   - `sbtc-withdrawal` does not adhere to any pattern followed by the other contracts.
   -
3. `sbtc-bootstrap-signers` includes `;;` placeholders in empty sections (in `data vars` and `data maps`), while while the other contracts do not use placeholders.
4. `sbtc-bootstrap-signers` has a 3 newline gap between functions, instead of just 1 newline. In some places, `sbtc-registry` has 2 newline gaps between headers, instead of one (`[1]` and `[2]`).
5. The `sbtc-token` contract uses tabs for indentation, while the other contracts use spaces.

Enhancing code uniformity and, in some cases, reducing contract size can be achieved by addressing these differences.

Recommendation

1. Standardize the use of uppercase for constants.
2. Adopt a uniform contract layout for all contracts. The layout of `sbtc-registry` appears to be more appropriate.
3. Remove the `;;` placeholder strings.
4. Ensure there is only one newline between code elements.
5. Use tabs for indentation in all contracts instead of spaces. This change would also reduce code size and implicitly lower runtime costs.

CONTENTS

ClarityAlliance
Security Review

sBTC

[QA-08] Use Constants Where Appropriate

Description

To enhance code readability, it is recommended to use meaningful constants where applicable. Below are instances within the current codebases where constants can be utilized, along with suggestions:

- In `sbtc-bootstrap-signers`:
  - At L88, replace `0xae` and the `CHECKMULTISIG` comment with the constant `OP_CHECKMULTISIG`.
  - Replace all instances of `u80` with `OP_N_BASE` or `OP_N`.
  - At L106, replace `0x14` with `ADDRESS_VERSION_MAINNET_MULTISIG` and `0x15` with `ADDRESS_VERSION_TESTNET_MULTISIG`.

Recommendation

Implement the suggested changes.

[QA-09] Misleading, Outdated, or Incomplete Comments in sBTC Contracts

Description

The codebase contains comments that are either misleading or outdated.

Instances:

- `sbtc-deposit.clar#L79`: The comment `(up to 1000)` should be updated `(up to 650)` because the `complete-deposits-wrapper` is limited to 650 elements.
- `sbtc-bootstrap-signers.clar#L116-L118`: The comment regarding the iterator's maximum size is incorrect. The total maximum size is 4352, calculated as 33 (public key length) * 128 (maximum number of keys) +128 (one-byte length prefix for each address).
- `sbtc-withdrawal.clar#L209`: The comment `;; Call into registry to confirm accepted withdrawal` should be changed to `to reject withdrawal` as it is used in the context of rejection.
- `sbtc-registry.clar#L115`: The function description states `returns the current signer set as a list of principals`, but the output is a tuple with data, not a list of principals. It should be updated to reflect the complete data it returns.
- `sbtc-registry.clar#L97`: The function documentation incorrectly mentions returning the fields of `withdrawal-sweeps` instead of the correct `completed-withdrawal-sweep`.
- `sbtc-registry.clar#L108-L109`: The `get-deposit-status` function documentation is incorrectly copied from the `get-completed-deposit` function.
- `sbtc-registry.clar#L249`: The `complete-deposit` function documentation incorrectly states `Store a new insert request`. It should be corrected to indicate that the function stores a finalized deposit request.

Recommendation

Address the mentioned instances as recommended above.

CONTENTS

[QA-10] sBTC Protocol Contract Type Can Be Updated With Arbitrary Data

Description

The `sbtc-registry` contract maintains two mappings: one for authorized principals to authorization types (`active-protocol-roles`) and another in reverse (`active-protocol-contracts`).

There are only three valid protocol contract types:

```
;; Protocol contract type
(define-constant governance-role 0x00)
(define-constant deposit-role 0x01)
(define-constant withdrawal-role 0x02)
```

The protocol type can be updated via the `sbtc-registry::update-protocol-contract` function. However, this function lacks validation for new values being added. Similarly, the `sbtc-bootstrap-signers::update-protocol-contract-wrapper` function, which invokes the registry update function, also does not perform any validation.

As a result, an incorrect contract type might mistakenly be passed and accepted as valid by the current implementation. Such instances should trigger a reversion to alert callers, rather than being accepted without validation. Due to the peculiar manner in which protocol contract validations are conducted, there are no other side effects.

Recommendation

Ensure that any new contract type passed to either `sbtc-registry::update-protocol-contract` or `sbtc-bootstrap-signers::update-protocol-contract-wrapper` is validated to be one of the existing, supported values.

ClarityAlliance
Security Review
sBTC

[QA-11] Redundant Protocol Mapping in sBTC Contracts

Description

The new authorization schema for the sBTC contracts includes unnecessary operations.

The implementation uses the `active-protocol-contracts` map to link protocol roles to contract principals and the `active-protocol-roles` map to do the reverse, associating contract principals with active roles.

The `active-protocol-roles` map is utilized solely in the `is-protocol-caller` function, where it is checked alongside `active-protocol-contracts`:

```
;; Verify that the contract-caller is a protocol contract
(asserts! (is-eq (some contract)
  (map-get? active-protocol-contracts contract-flag)) ERR_UNAUTHORIZED)
;; Verify that the flag matches the contract-caller
(asserts! (is-eq (some contract-flag)
  (map-get? active-protocol-roles contract)) ERR_UNAUTHORIZED)
```

Since the principal-to-role association updates both `active-protocol-contracts` and `active-protocol-roles` simultaneously in `sbtc-registry::update-protocol-contract`, there will never be a situation where a principal-to-role is correctly set in one map but incorrectly in the other.

Recommendation

The `active-protocol-roles` map is redundant and should be removed to lower execution costs.

[QA-12] Some sBTC Protocol Setter Functions Lack Corresponding Role

Description

The new sBTC authorization mechanism mandates that only one contract can hold any given role at a time. Currently, there are only three roles defined:

```
;; Protocol contract type
(define-constant governance-role 0x00)
(define-constant deposit-role 0x01)
(define-constant withdrawal-role 0x02)
```

Upon mapping all protocol functions to their respective calls, it becomes evident that the `sbtc-token` functions `protocol-set-name`, `protocol-set-symbol` and `protocol-mint-many` lack associated roles that can execute them. Meanwhile, the three existing roles are already assigned:

```
(map-set active-protocol-contracts governance-role .sbtc-bootstrap-signers)
(map-set active-protocol-contracts deposit-role .sbtc-deposit)
(map-set active-protocol-contracts withdrawal-role .sbtc-withdrawal)
```

As a result, if the sBTC team and governance decide to modify any metadata on the sBTC contract (although changing the name and symbol is not recommended and is discussed in another issue), they must call `sbtc-bootstrap-signers::update-protocol-contract-wrapper` with an arbitrary role that does not conflict with the existing roles (0×00 - 0×02). This role must be assigned to a different principal than any of the current three contracts (e.g., assigning it to `current-signer-principal` itself) before calling the `sbtc-token` functions.

The absence of designated roles for these functions necessitates a workaround of the existing authorization mechanism by the development team.

Recommendation

Establish a metadata role to manage the `protocol-set-name` and `protocol-set-symbol` functions, and a minter role to handle the `protocol-mint-many` and `protocol-mint` functions. If multiple contracts per role are needed, consider removing the current limitation of the authorization mechanism that restricts each role to a single principal.

CONTENTS

[QA-13] Outdated Emily API Domain Name Generation Schema Documentation for Non-Production Environments

Description

In the Emily TypeScript CDK deployment scripts, when configuring the APIs via the `emily-stack::createOrUpdateSpecificApi` functions, an Amazon Route 53 DNS is used to set up specific domains for the APIs.

The custom domain varies depending on whether the deployment stage is production or not, as indicated in the comments:

```
// Create the custom domain name of the format:
// if stage != prod: [stage].[purpose].[customRootDomainNameRoot]
// if stage == prod: [purpose].[customRootDomainNameRoot]
```

However, the actual implementation incorrectly swaps the stage with purpose placeholders for non-production environments:

```
const
    customDomainName = `${purposePrefix}${stagePrefix}${customRootDomainNameRoot}`;
```

This results in the domain ending as `[purpose].[stage].[customRootDomainNameRoot]`, which is actually the intended design due to a DNS limitation explained in PR#1112.

Recommendation

In the `emily-stack::createOrUpdateSpecificApi` function, update the custom domain format comments to reflect the current design accurately.

CONTENTS

[QA-14] Simplification of EmilyStackUtils Operations

Description

The `emily-stack-utils::EmilyStackUtils` class offers utility methods for the Cloud Formation Stack. There are two functions with redundant operations that can be streamlined:

- The `isDevelopmentStack` function, in the worst-case scenario, calls the (non-cached) `getStageName` function four times.
- In the `getLambdaGitIdentifier` function, the branch `throw new Error('Failed to get the git identifier for the lambda.');` is not reachable.

By simplifying or removing redundant code, the codebase becomes easier to maintain.

Recommendation

Revise the `isDevelopmentStack` function to make a single call to `isDevelopmentStack`. An example implementation is as follows:

```
public static isDevelopmentStack(): boolean {
    return [
        Constants.DEV_STAGE_NAME,
        Constants.LOCAL_STAGE_NAME,
        Constants.UNIT_TEST_STAGE_NAME,
        Constants.TEMP_STAGE_NAME
    ].includes(this.getStageName());
}
```

In the `getLambdaGitIdentifier` function, remove the second check for `this.lambdaGitIdentifier` being equal to `undefined`.

CONTENTS

ClarityAlliance
Security Review
sBTC

[QA-15] Cleanup EmilyStack Class

Description

The `emily-stack::EmilyStack` class has several areas that would benefit from a code cleanup:

1. **Remove Unused Imports**

```
import * as logs from "aws-cdk-lib/aws-logs";
import * as cr from "aws-cdk-lib/custom-resources";
```

2. The result of calling `createOrUpdateApi` is stored in the `emilyApis` constant, which is never used. The function can be called without saving the return value.

3. The JSDoc `@param` tags for the `createOrUpdateDepositTable` function indicate `tableId` and `tableName` parameters, while the actual parameters are `depositTableId` and `depositTableName` Ensure the JSDoc matches the actual implementation.

4. DynamoDB automatically projects the primary key attributes of the base table into all Global Secondary Indexes (GSIs) and Local Secondary Indexes (LSIs). Therefore, adding them again in the `nonKeyAttributes` of the secondary index is redundant.

For example, adding `BitcoinTxid` (partition key) and `BitcoinTxOutputIndex` (sort key) to the secondary index `nonKeyAttributes` is unnecessary. For the `withdrawals` and `deposits` tables, remove the primary key attributes from the secondary index `nonKeyAttributes` field.

If duplicating the fields is intentional, add a comment to indicate that they are redundantly added so that no developer incorrectly assumes their indexing is optional or dependent on their presence in the `nonKeyAttributes` entry.

Recommendation

Implement the suggested code alterations.

ClarityAlliance
Security Review
sBTC

[QA-16] Typographical Errors in Emily

Description

The Emily API codebase contains several typographical errors:

- In `emily-stack-utils`:
  - `apiJsonDefiniton` should be corrected to `apiJsonDefinition`
  - `exercize` should be corrected to `exercise`
- In `handler\src\api\handlers\internal.rs`:
  - `occured` should be corrected to `occurred`
  - `intendeded` should be corrected to `intended`
- In `handler\src\api\models\common\mod.rs` and `handler\src\database\entries\mod.rs`:
  - `transaciton` should be corrected to `transaction`
  - `articacts` should be corrected to `artifacts`
  - `fulill` should be corrected to `fulfill`
- In `handler\src\api\models\deposit\requests.rs` and `handler\src\api\models\withdrawal\requests.rs`:
  - `singlular` should be corrected to `singular`
- In `handler\src\api\routes\mod.rs`:
  - `definitiions` should be corrected to `definitions`
- In `handler\src\common\error.rs`:
  - The `Reorganzing` error should be renamed to `Reorganizing`
- In `handler\src\database\entries\deposit.rs` and `handler\src\database\entries\withdrawal.rs`:
  - `chronoloical` should be corrected to `chronological`
  - `chainsates` should be corrected to `chainstates`
- `handler\src\database\entries\deposit.rs`:
  - `depoit` should be corrected to `deposit`
- In `handler\src\database\entries\mod.rs`:
  - `Parition` should be corrected to `Partition`
  - `serialied` should be corrected to `serialized`
- In `handler\src\database\accessors.rs`:
  - `exhasutive` should be corrected to `exhaustive`
  - `sigular` should be corrected to `singular`
  - `parition_key` should be corrected to `partition_key`

Recommendation

Correct all identified typographical errors.

CONTENTS

ClarityAlliance
Security Review

sBTC

[QA-17] Use Constants Instead of Magic Numbers in Emily APIs

Description

In the Emily TypeScript CDK deployment scripts, several key values are directly hardcoded into the code instead of being moved to the existing `Constants` class.

- In `emily-stack.ts#L309-L310`, the lambda timeout of 5 seconds should be moved to a constant.
- In `emily-stack.ts#L413-L415`, the request limits (rate and burst) should be moved into separate constants.
- In `chainstate.rs#L189`, `deposit.rs#L363`, and `withdrawal.rs#L253`, the retry count for database updates is hardcoded as 15 and should be set as a constant.

Generally, using constants instead of hardcoding values improves code readability and simplifies maintenance when configuration changes are needed.

Recommendation

Implement constants in the specified instances.