BLOCKSEC

Security Audit Report for Bridge Contract

Date: April 1, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Pacifica
Target	Bridge Contract

Version History

Version	Date	Description
1.0	April 1, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-07 07:52:57+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Bridge Contract of Pacifica.

Bridge Contract is a cross-chain bridge protocol on Solana designed to facilitate transfers of SOL and SPL tokens across different chains. The protocol operates by emitting on-chain logs that are observed and processed by an off-chain relayer service, which coordinates and executes corresponding actions on the destination chain. The system is composed of two dedicated programs: sol-bridge, which handles native SOL bridging, and spl-bridge, which manages SPL token registration and transfers. This separation allows for clearer responsibility boundaries and more flexible handling of different asset types within the bridging workflow.

Note this audit only focuses on the smart contracts in the following directories/files:

- programs/sol-bridge
- programs/spl-bridge

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
Bridge Contract	`Version 1`	`15d0aa7a4269beba47f67f9af35eb7d85525adaf`
	`Version 1`	`4dd0ff83094a5cfb61856444953356d50e62ee40`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/pacifica-fi/bridge-contract

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism

* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **two** recommendations and **two** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 2
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Potential fund lock due to lack of check in function `remove_token()`	Security Issue	Fixed
2	Low	Lack of `DefaultAccountState` extension check	Security Issue	Fixed
3	Low	Lack of minimum deposit amount validation	Security Issue	Confirmed
4	-	Reject empty `withdrawals` in function `batch_withdraw()`	Recommendation	Fixed
5	-	Improper check in function `batch_withdraw()`	Recommendation	Fixed
6	-	Potential centralization risks	Note	-
7	-	Ensure the off-chain event monitoring matches on-chain event definitions	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential fund lock due to lack of check in function `remove_token()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `remove_token()` removes a supported token by setting `supported_token.is_active = false`. The function `batch_withdraw()` can only withdraw tokens that are currently supported. If a token is removed while assets still remain in `bridge_token_account`, function `batch_withdraw()` can no longer be used to withdraw them. Since the protocol does not support resetting `supported_token.is_active` to true, the assets will be permanently locked in `bridge_token_account`.

```
34   impl RemoveToken<'_> {
35       pub fn handle(ctx: Context<RemoveToken>) -> Result<()> {
36           let clock = Clock::get()?;
37
38           let supported_token = &mut ctx.accounts.supported_token;
39           supported_token.is_active = false;
40
```

```
41          emit_cpi!(TokenRemoved {
42              mint: ctx.accounts.mint.key(),
43              timestamp: clock.unix_timestamp,
44          });
45
46          Ok(())
47      }
48  }
```

Listing 2.1: programs/spl-bridge/src/instructions/remove_token.rs

Impact Any tokens that remain in `bridge_token_account` after `remove_token` is invoked are permanently irrecoverable absent a program upgrade.

Suggestion Revise the logic accordingly.

2.1.2 Lack of `DefaultAccountState` extension check

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The protocol allows privileged accounts to register supported tokens via function `add_token()`. However, the function `add_token()` does not check for certain Token-2022 extensions, such as `DefaultAccountState`.

If a mint is configured with `DefaultAccountState::Frozen`, Anchor's `init` will initialize `bridge_token_account` and `fallback_token_account` in a frozen state, they will be unable to receive tokens.

```
74  impl AddToken<'_> {
75      pub fn handle(ctx: Context<AddToken>) -> Result<()> {
76          // Check for dangerous Token-2022 extensions
77          if ctx.accounts.token_program.key() == spl_token_2022::ID {
78              let mint_account_info = ctx.accounts.mint.to_account_info();
79              let mint_data = mint_account_info.try_borrow_data()?;
80              let mint_state = StateWithExtensions::<Mint2022>::unpack(&mint_data)
81                  .map_err(|_| BridgeError::UnsupportedTokenExtension)?;
82
83              // Reject tokens with dangerous extensions
84              require!(
85                  mint_state.get_extension::<NonTransferable>().is_err(),
86                  BridgeError::UnsupportedTokenExtension
87              );
88              require!(
89                  mint_state.get_extension::<PermanentDelegate>().is_err(),
90                  BridgeError::UnsupportedTokenExtension
91              );
92              require!(
93                  mint_state
94                      .get_extension::<ConfidentialTransferMint>()
95                      .is_err(),
96                  BridgeError::UnsupportedTokenExtension
```

```
97              );
98              require!(
99                  mint_state.get_extension::<TransferHook>().is_err(),
100                 BridgeError::UnsupportedTokenExtension
101             );
102             // Transfer fees would cause accounting issues - deposited amount != withdrawn
                    amount
103             require!(
104                 mint_state.get_extension::<TransferFeeConfig>().is_err(),
105                 BridgeError::UnsupportedTokenExtension
106             );
107         }
108     }
109 }
```

Listing 2.2: programs/spl-bridge/src/instructions/add_token.rs

Impact The frozen `bridge_token_account` and `fallback_token_account` will not be able to receive tokens.

Suggestion Add the `DefaultAccountState` extension check.

2.1.3 Lack of minimum deposit amount validation

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The program `sol-bridge` facilitates native SOL transfers from users to the bridge vault through the function `handle()`. The function `handle()` does not implement a minimum deposit check that synchronizes with the off-chain processing logic. This inconsistency causes the state manager to consume the deposit nonce without crediting funds to the user, which leads to the irreversible loss of small-value deposits.

The program `spl-bridge` has a similar issue.

```
30  impl DepositSol<'_> {
31      pub fn handle(ctx: Context<DepositSol>, recipient: Pubkey, amount: u64) -> Result<()> {
32          require!(amount > 0, BridgeError::ZeroAmount);
33          require!(
34              recipient != Pubkey::default(),
35              BridgeError::InvalidRecipient
36          );
37
38          let clock = Clock::get()?;
39
40          system_program::transfer(
41              CpiContext::new(
42                  ctx.accounts.system_program.to_account_info(),
43                  system_program::Transfer {
44                      from: ctx.accounts.depositor.to_account_info(),
45                      to: ctx.accounts.bridge_authority.to_account_info(),
46                  },
```

```
47              ),
48              amount,
49          )?;
50
51          let deposit_nonce = ctx.accounts.bridge_config.deposit_nonce;
52
53          emit_cpi!(Deposit {
54              source: ctx.accounts.depositor.key(),
55              recipient,
56              mint: Pubkey::default(),
57              amount,
58              deposit_nonce,
59              timestamp: clock.unix_timestamp,
60          });
61
62          ctx.accounts.bridge_config.deposit_nonce =
63              deposit_nonce.checked_add(1).ok_or(BridgeError::Overflow)?;
64
65          Ok(())
66      }
67  }
```

Listing 2.3: programs/sol-bridge/src/instructions/deposit_sol.rs

Impact Off-chain systems will consume deposit nonces without updating account balances, which may leads to the irreversible loss of small-value deposits.

Suggestion Revise the logic accordingly.

Feedback from the project The team confirmed that irreversible loss of small-value deposits is an intentional design choice. Minimum deposit amounts adjust based on asset prices. For simplification, this dynamic logic and enforcement of minimum deposit amounts is only tracked on the backend. Minimum deposit amounts will be clearly documented so users are aware of this irreversible loss behavior.

2.2 Recommendation

2.2.1 Reject empty `withdrawals` in function `batch_withdraw()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the programs `spl-bridge` and `sol-bridge`, the function `batch_withdraw()` releases bridged assets by iterating over the input `withdrawals` array and transferring funds to each recipient. However, the function does not verify that the input `withdrawals` array is non-empty. If an empty array is provided, no transfer is executed; nevertheless, the function still emits the event `BatchWithdrawalCompleted` and increments the variable `withdraw_nonce`, which is operationally meaningless and may cause unnecessary nonce consumption and misleading off-chain accounting.

```
40  pub fn batch_withdraw<'info>(
```

```
41      ctx: Context<'_, '_, 'info, 'info, BatchWithdraw<'info>>,
42      nonce: u64,
43      withdrawals: Vec<WithdrawalRequest>,
44  ) -> Result<()> {
45      BatchWithdraw::handle(ctx, nonce, withdrawals)
46  }
```

Listing 2.4: programs/sol-bridge/src/lib.rs

```
42  pub fn handle(
43      ctx: Context<'_, '_, 'info, 'info, BatchWithdraw<'info>>,
44      nonce: u64,
45      withdrawals: Vec<WithdrawalRequest>,
46  ) -> Result<()> {
47      require!(
48          nonce == ctx.accounts.bridge_config.withdraw_nonce,
49          BridgeError::InvalidNonce
50      );
51
52      require!(
53          ctx.remaining_accounts.len() == withdrawals.len(),
54          BridgeError::Unauthorized
55      );
56
57      let clock = Clock::get()?;
58      let bridge_authority_bump = ctx.bumps.bridge_authority;
59      let authority_seeds: &[&[u8]] = &[BRIDGE_AUTHORITY, &[bridge_authority_bump]];
60
61      let mut successful_withdrawals: Vec<WithdrawalRecord> =
62          Vec::with_capacity(withdrawals.len());
63
64      for (idx, withdrawal) in withdrawals.iter().enumerate() {
65          require!(withdrawal.amount > 0, BridgeError::ZeroAmount);
66          require!(
67              withdrawal.recipient != Pubkey::default(),
68              BridgeError::InvalidRecipient
69          );
70
71          let recipient_info = &ctx.remaining_accounts[idx];
72
73          require!(
74              recipient_info.key() == withdrawal.recipient,
75              BridgeError::InvalidRecipient
76          );
77
78          let bridge_balance = ctx.accounts.bridge_authority.lamports();
79          require!(
80              bridge_balance >= withdrawal.amount,
81              BridgeError::InsufficientBridgeBalance
82          );
83
84          let recipient_lamports = recipient_info.lamports();
85
```

```
86          if recipient_lamports == 0 {
87              system_program::transfer(
88                  CpiContext::new_with_signer(
89                      ctx.accounts.system_program.to_account_info(),
90                      system_program::Transfer {
91                          from: ctx.accounts.bridge_authority.to_account_info(),
92                          to: ctx.accounts.fallback_authority.to_account_info(),
93                      },
94                      &[authority_seeds],
95                  ),
96                  withdrawal.amount,
97              )?;
98
99              emit_cpi!(WithdrawalToFallback {
100                 recipient: withdrawal.recipient,
101                 mint: Pubkey::default(),
102                 amount: withdrawal.amount,
103                 reason: "Recipient account does not exist (0 lamports)".to_string(),
104                 timestamp: clock.unix_timestamp,
105             });
106         } else {
107             let transfer_result = system_program::transfer(
108                 CpiContext::new_with_signer(
109                     ctx.accounts.system_program.to_account_info(),
110                     system_program::Transfer {
111                         from: ctx.accounts.bridge_authority.to_account_info(),
112                         to: recipient_info.clone(),
113                     },
114                     &[authority_seeds],
115                 ),
116                 withdrawal.amount,
117             );
118
119             match transfer_result {
120                 Ok(_) => {
121                     successful_withdrawals.push(WithdrawalRecord {
122                         recipient: withdrawal.recipient,
123                         amount: withdrawal.amount,
124                     });
125                 }
126                 Err(e) => {
127                     system_program::transfer(
128                         CpiContext::new_with_signer(
129                             ctx.accounts.system_program.to_account_info(),
130                             system_program::Transfer {
131                                 from: ctx.accounts.bridge_authority.to_account_info(),
132                                 to: ctx.accounts.fallback_authority.to_account_info(),
133                             },
134                             &[authority_seeds],
135                         ),
136                         withdrawal.amount,
137                     )?;
138
```

```
                    emit_cpi!(WithdrawalToFallback {
                        recipient: withdrawal.recipient,
                        mint: Pubkey::default(),
                        amount: withdrawal.amount,
                        reason: format!("Transfer failed: {}", e),
                        timestamp: clock.unix_timestamp,
                    });
                }
            }
        }
    }

    emit_cpi!(BatchWithdrawalCompleted {
        nonce,
        mint: Pubkey::default(),
        withdrawer: ctx.accounts.withdrawer.key(),
        withdrawals: successful_withdrawals,
        timestamp: clock.unix_timestamp,
    });

    ctx.accounts.bridge_config.withdraw_nonce =
        nonce.checked_add(1).ok_or(BridgeError::Overflow)?;

    Ok(())
}
```

Listing 2.5: programs/sol-bridge/src/instructions/batch_withdraw.rs

```
pub fn batch_withdraw<'info>(
    ctx: Context<'_, '_, 'info, 'info, BatchWithdraw<'info>>,
    mint: Pubkey,
    nonce: u64,
    withdrawals: Vec<WithdrawalRequest>,
) -> Result<()> {
    BatchWithdraw::handle(ctx, mint, nonce, withdrawals)
}
```

Listing 2.6: programs/spl-bridge/src/lib.rs

```
pub fn handle(
    ctx: Context<'_, '_, 'info, 'info, BatchWithdraw<'info>>,
    mint: Pubkey,
    nonce: u64,
    withdrawals: Vec<WithdrawalRequest>,
) -> Result<()> {
    // Validate nonce matches expected value
    require!(
        nonce == ctx.accounts.supported_token.withdraw_nonce,
        BridgeError::InvalidNonce
    );

    // Validate remaining accounts length
    require!(
```

```
92          ctx.remaining_accounts.len() == withdrawals.len(),
93          BridgeError::Unauthorized
94      );
95
96      let clock = Clock::get()?;
97      let bridge_authority_bump = ctx.bumps.bridge_authority;
98      let authority_seeds: &[&[u8]] = &[BRIDGE_AUTHORITY, &[bridge_authority_bump]];
99
100     let decimals = ctx.accounts.mint_account.decimals;
101
102     // Track successful withdrawals for batch event
103     let mut successful_withdrawals: Vec<WithdrawalRecord> =
104         Vec::with_capacity(withdrawals.len());
105
106     // Process each withdrawal
107     for (idx, withdrawal) in withdrawals.iter().enumerate() {
108         require!(withdrawal.amount > 0, BridgeError::ZeroAmount);
109         require!(
110             withdrawal.recipient != Pubkey::default(),
111             BridgeError::InvalidRecipient
112         );
113
114         // Get recipient token account from remaining_accounts
115         let recipient_token_info = &ctx.remaining_accounts[idx];
116
117         // Calculate expected ATA on-chain (supports both legacy SPL and Token-2022)
118         let expected_ata =
119             anchor_spl::associated_token::get_associated_token_address_with_program_id(
120                 &withdrawal.recipient,
121                 &mint,
122                 &ctx.accounts.token_program.key(),
123             );
124
125         require!(
126             recipient_token_info.key() == expected_ata,
127             BridgeError::InvalidRecipient
128         );
129
130         // Check if ATA is initialized (account has data)
131         let should_use_fallback = if recipient_token_info.data_is_empty() {
132             Some("ATA not initialized")
133         } else {
134             // Parse the token account to check for frozen state and CPI guard
135             let account_data = recipient_token_info.try_borrow_data()?;
136
137             // Check for Token-2022 specific extensions
138             if ctx.accounts.token_program.key() == spl_token_2022::ID {
139                 if let Ok(account_state) =
140                     StateWithExtensions::<Account2022>::unpack(&account_data)
141                 {
142                     // Check if account is frozen
143                     if account_state.base.is_frozen() {
144                         Some("Account frozen")
```

```
                    }
                    // Check if CPI Guard is enabled
                    else if account_state
                        .get_extension::<CpiGuard>()
                        .map(|g| g.lock_cpi.into())
                        .unwrap_or(false)
                    {
                        Some("CPI Guard enabled")
                    } else {
                        None
                    }
                } else {
                    Some("Failed to parse account")
                }
            } else {
                // Legacy SPL token - just check frozen state
                // TokenAccount is 165 bytes, frozen flag is at offset 108
                if account_data.len() >= 109 && account_data[108] == 2 {
                    // AccountState::Frozen = 2
                    Some("Account frozen")
                } else {
                    None
                }
            }
        };

        if let Some(reason) = should_use_fallback {
            // Transfer to fallback
            transfer_checked(
                CpiContext::new_with_signer(
                    ctx.accounts.token_program.to_account_info(),
                    TransferChecked {
                        from: ctx.accounts.bridge_token_account.to_account_info(),
                        mint: ctx.accounts.mint_account.to_account_info(),
                        to: ctx.accounts.fallback_token_account.to_account_info(),
                        authority: ctx.accounts.bridge_authority.to_account_info(),
                    },
                    &[authority_seeds],
                ),
                withdrawal.amount,
                decimals,
            )?;

            // Emit fallback event
            emit_cpi!(WithdrawalToFallback {
                recipient: withdrawal.recipient,
                mint,
                amount: withdrawal.amount,
                reason: reason.to_string(),
                timestamp: clock.unix_timestamp,
            });
        } else {
            // Transfer to recipient
```

```
198             transfer_checked(
199                 CpiContext::new_with_signer(
200                     ctx.accounts.token_program.to_account_info(),
201                     TransferChecked {
202                         from: ctx.accounts.bridge_token_account.to_account_info(),
203                         mint: ctx.accounts.mint_account.to_account_info(),
204                         to: recipient_token_info.clone(),
205                         authority: ctx.accounts.bridge_authority.to_account_info(),
206                     },
207                     &[authority_seeds],
208                 ),
209                 withdrawal.amount,
210                 decimals,
211             )?;
212 
213             // Track successful withdrawal
214             successful_withdrawals.push(WithdrawalRecord {
215                 recipient: withdrawal.recipient,
216                 amount: withdrawal.amount,
217             });
218         }
219     }
220 
221     // Emit batch completion event with all successful withdrawals
222     emit_cpi!(BatchWithdrawalCompleted {
223         nonce,
224         mint,
225         withdrawer: ctx.accounts.withdrawer.key(),
226         withdrawals: successful_withdrawals,
227         timestamp: clock.unix_timestamp,
228     });
229 
230     // Increment withdraw nonce as LAST operation
231     ctx.accounts.supported_token.withdraw_nonce =
232         nonce.checked_add(1).ok_or(BridgeError::Overflow)?;
233 
234     Ok(())
235 }
```

Listing 2.7: programs/spl-bridge/src/instructions/batch_withdraw.rs

Suggestion Add a check to ensure the input `withdrawals` array is not empty.

2.2.2 Improper check in function `batch_withdraw()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the legacy `SPL` path, `BatchWithdraw` infers frozen state by reading `account_data[108] when len >= 109`. While this works for standard `TokenAccount` layouts, it is only a partial structural check and may lose the required error message.

```
159             else {
```

```
160                 // Legacy SPL token - just check frozen state
161                 // TokenAccount is 165 bytes, frozen flag is at offset 108
162                 if account_data.len() >= 109 && account_data[108] == 2 {
163                     // AccountState::Frozen = 2
164                     Some("Account frozen")
165                 } else {
166                     None
167                 }
168             }
```

Listing 2.8: programs/spl-bridge/src/instructions/batch_withdraw.rs

Suggestion Use `spl_token::state::Account::unpack(&account_data)` and treat `Err(_)` as a fallback trigger.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `withdrawer`) can conduct sensitive operations, which introduces potential centralization risks. For example, a `withdrawer` is able to transfer funds held by the `fallback_authority` to any arbitrary address. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Ensure the off-chain event monitoring matches on-chain event definitions

Introduced by `Version 1`

Description In this project, the cross-chain mechanism relies on off-chain listeners to detect and process on-chain events. It is important to ensure that the off-chain logic is implemented correctly and remains fully consistent with the on-chain logic, including event formats, parameter encoding, and processing semantics. Any mismatch between off-chain and on-chain behavior may lead to incorrect execution or unexpected states in the bridging workflow.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS