Self Chain

Blockchain Security Audit

No. 202404191527

Apr 19th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Result

After auditing, 5 Medium-risks ,4 Low-risks and 3 Info items were identified in the Self Chain. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Medium	Fixed : 4	Acknowledged: 1
Low	Fixed : 4	Acknowledged: 0
Info	Fixed : 2	Acknowledged: 1

Business overview

The Self Chain is a Layer1 blockchain built on Cosmos. In addition to the basic modules of Cosmos, Self Chain also adds two modules: migration and selfvesting to provide users with the function of migrating Ethereum assets to Self Chain. In addition, in order to automate the migration process, a migration contract `DeadWallet` and a migration service are also built: `selfchain -migrator`.

The `DeadWallet` contract is a smart contract on the Ethereum. It provides an entrance for migrating tokens to self-chains.Users can lock FRONT and HOTCROSS tokens into this contract, and the contract will automatically trigger relevant events, which will be received by `selfchain-migrator`. After `selfchain-migrator` monitors the migration request from the `DeadWallet` contract, it will perform a simple detection, and then call the interface of the migration module in Self Chain to create a vesting record. After the vesting duration is over, users can call the release function of the self-unlocking module to release this part of the assets.

In addition to the above-mentioned automated migration, Self Chain also provides the function for project manager to manually add migration records to avoid missing migration records due to network problems.

1 Overview

1.1 Project Overview

Project Name	Self Chain
Project Language	Solidity, Go, Rust
Platform	Self Chain
Code Base	https://github.com/hotcrosscom/selfchain https://github.com/hotcrosscom/selfchain-migrator https://github.com/hotcrosscom/token-migration
Commit ID	selfchain: bb87ac2289f71759a642a9c9c7ce00a84c7accfd 02ba28d8c08a784891982c49aeec23ec10407a36 14640648e19727626e62b21a8563bcbb839907b1 1c7c71f5949f94132c03c6abb7dcc9bbdf1da938 a489c4198f14afd26666588ca714e645bf19694a selfchain-migrator: e0d2d421fc6fdc92ace874d1befe05402df426f8 fa02c0fa22004078cd101d7126971c365e7e302b 6858e50ec61cd774f311e85e74920da53b6cfe12 b9c0943e48e4e3731983c6e0d4e21ded18a932cf token-migration: 99c22b52afbb898e41bcca4d47679b83027a605e 7ae833df9779af2787bbb302c8050c7390cac2c6

1.2 Audit Overview

Audit work duration: Mar 12, 2024 – Apr 19, 2024

Audit team: Beosin Security Team

2 Findings

Index	Risk description	Severity level	Status
Self Chain-01	Periodic vesting accounts lack validation	Medium	Fixed
Self Chain-02	Excessive broadcast failures can result in user funds being locked	Medium	Acknowledged
Self Chain-03	Migration request status processing is incomplete	Medium	Fixed
Self Chain-04	Improper concurrency operations may result in data loss	Medium	Fixed
Self Chain-05	DeadWallet contract missing check for migration amount	Medium	Fixed
Self Chain-06	Potential slashing evasion during re-delegation	Low	Fixed
Self Chain-07	The lockedAmount missing numeric check	Low	Fixed
Self Chain-08	Transaction failed due to wrong token type	Low	Fixed
Self Chain-09	Missing BlockedAddressed validation in vesting module	Low	Fixed
Self Chain-10	The target address in the DeadWallet contract is not verified	Info	Acknowledged
Self Chain-11	Redundant code	Info	Fixed
Self Chain-12	Query function is missing parameter restrictions	Info	Fixed

[Self Chain-01] Periodic vesting accounts lack validation

Severity Level Medium

Lines https://github.com/cosmos/cosmos-sdk/tree/v0.46.7/x/auth/vesting/types/msgs.go #L163-191

Description The Self Chain utilizes `Cosmos-SDK v0.46.7`, which contains a security vulnerability. Specifically, in this version, the `PeriodicVestingAccount` lacks proper validation for the corresponding vesting period. If the amount within the vesting is invalid, it allows deposits but does not permit withdrawals. Consequently, when a user deposits funds into their account, those funds become permanently locked, and the user is unable to withdraw them.

```
    if msg.StartTime < 1 {
        return fmt.Errorf("invalid start time of %d, length must be greater than 0", msg.StartTime)
    }
    for i, period := range msg.VestingPeriods {
        if period.Length < 1 {
            return fmt.Errorf("invalid period length of %d in period %d, length must be greater than 0", period.Length, i)
        }
    }
```

Recommendation It is recommended to add additional period validation logic or upgrade the SDK version to v0.46.13 or higher.

Reference:
https://github.com/cosmos/cosmos-sdk/commit/fd90480b0a922611e366552751a9037e309d8410

Status **Fixed.** The issue has been fixed in commit 02ba28d8c08a784891982c49aeec23ec10407a36 of the project. The current version of `Cosmos-SDK` being used is 0.47.10.

[Self Chain-02] Excessive broadcast failures can result in user funds being locked

Severity Level	Medium
Lines	selfchain-migrator/src/consumers/migration_request.rs
Description	In selfchain-migrator/src/consumers/migration_request.rs, the `MigrationRequestConsumer` handler lacks error handling mechanisms, potentially resulting in failed migration operations for users. For instance, if network or other issues prevent the successful broadcast of a mint transaction on the Self Chain, the handler retries the operation. However, after 50 retry attempts, regardless of the success or failure of the Self Chain's migration transaction, the program proceeds to consume messages from the RabbitMQ message queue and marks the corresponding user transaction as processed. This situation can lead to funds being locked and migration failures for users.
Recommendation	It is recommended to, after reaching the maximum retries, store the information of the failed transaction in a database for manual intervention to perform a manual migration at a later stage instead of directly deleting migration records.
Status	**Acknowledged.** The project team stated that the migration records have been recorded through logs and no longer need to be stored separately.

```
    if retry_count > 50 {
      println!("Failed to send migrate tokens for tx hash {:?}",
&msg.tx_hash);
      return Ok(())
    }
```

[Self Chain-03] Migration request status processing is incomplet

Severity Level	**Medium**
Lines	selfchain-migrator/src/blockchain/api.rs #L52-60
Description	In the api.rs of selfchain-migrator/src/blockchain, if a non-404 error occurs due to network or server problems, true will also be returned here, causing the migration request message of the message queue to be consumed, and the migration program mistakenly believes that it has been migrated. Even if the service is restored, the migration program will not reprocess the request, resulting in the user not minting the corresponding assets on Self Chain.

```
pub async fn migration_exists(api_url: &str, req: &MigrationRequest)
-> Result<bool> {
  let resp = fetch_token_migration(api_url, req).await?;
  if resp.status().as_u16() == 404 {
    Ok(false)
  } else {
    Ok(true)
  }
}
```

Recommendation	It is recommended to modify the matching conditions as follows: If it is a 404 error, it means "Not Migrated". If it is a 200 status code, it means "Migrated". Otherwise, if it is any other status code, return an error and resend the request.
Status	**Fixed.** This issue has been fixed according to the modification recommendations.

```
pub async fn migration_exists(api_url: &str, req: &MigrationRequest)
-> Result<bool> {
  let resp = fetch_token_migration(api_url, req).await?;
  let status = resp.status().as_u16();

  if status == 200 {
    Ok(true)
  } else if status == 404 {
    Ok(false)
  } else {
    Err(eyre!("Network error: {}", status))
  }
}
```

[Self Chain-04] Improper concurrency operations may result in data loss

Severity Level	Medium
Lines	selfchain-migrator/src/services/migration_listener.rs
Description	In selfchain-migrator/src/services/migration_listener.rs, the program first call `update_block(&store, Arc::clone(&shared_lock))` to asynchronously attempt updating Redis data every 300 seconds, then proceeds to call `read_block(&store.redis_pool).await` to read the Redis data. If there's a temporary failure in reading due to a Redis service issue, it awaits until the asynchronous sleep of 300 seconds completes. If during this wait period the Redis data is updated first, and then read, it results in fetching the latest block, potentially causing the loss of previously unprocessed blocks.

```
  let shared_lock = Arc::new(Mutex::new(()));
  update_block(&store, Arc::clone(&shared_lock));
  let start_block = if let Ok(start_block) =
read_block(&store.redis_pool).await {
    start_block
  } else {
    store.config.start_block
  };
```

Recommendation	It is recommended to read the data before performing write operations in asynchronous operations to avoid reading stale or incorrect data. In this case, consider placing the `update_block()` operation after the `read_block()` operation.
Status	**Fixed.** This issue has been fixed according to the modification recommendations.

```
  let start_block = if let Ok(start_block) =
read_block(&store.redis_pool).await {
    start_block
  } else {
    store.config.start_block
  };
  let shared_lock = Arc::new(Mutex::new(()));
  update_block(&store, Arc::clone(&shared_lock));
```

[Self Chain-05] DeadWallet contract missing check for migration amount

Severity Level	Medium
Lines	token-migration/contracts/DeadWallet.sol

Description	The `DeadWallet` contract serves as an Ethereum contract for receiving cross-chain assets. Users utilize the `migrateFront` and `migrateHotcross` functions in the contract to lock their assets. However, these two functions do not verify the amount of tokens being migrated. Meanwhile, Self Chain mandates that the amount of tokens being locked must exceed `config.MinMigrationAmount` during migration operations. Consequently, if a user's migrated token amount is less than this value, the migration will fail, leaving the user unable to withdraw their locked assets, resulting in asset loss.
	`amount := sdkmath.NewUintFromString(msg.Amount)`
	`if amount.LT(sdkmath.NewUint(config.MinMigrationAmount)) {`
	`    return nil, types.ErrInvalidMigrationAmount`
	`}`

Recommendation	It is recommended to check the user balance in the `migrateFront` and `migrateHotcross` functions, requiring the user balance to be no less than 1 token.
Status	**Fixed.** (code below)

```
    amount := sdkmath.NewUintFromString(msg.Amount)
    if amount.LT(sdkmath.NewUint(config.MinMigrationAmount)) {
        return nil, types.ErrInvalidMigrationAmount
    }
```

Fixed.

```
  uint256 constant MinMigrationAmount = 1e18;

  function setMigrationWindowOpen(uint256 token, bool isOpen) public
onlyOwner {
    if(token == FRONT_TOKEN) {
      isFrontOpen = isOpen;
    } else if (token == HOTCROSS_TOKEN) {
      isHotcrossOpen = isOpen;
    }
  }

  function migrateFront(string memory destAddress)
whenOpen(FRONT_TOKEN) public {
    uint256 amount = front.balanceOf(msg.sender);
    require(amount >= MinMigrationAmount, "Insufficient FRONT
balance");
    front.safeTransferFrom(msg.sender, address(this), amount);
    emit NewMigration(msg.sender, FRONT_TOKEN, destAddress, amount);
  }
```

[Self Chain-06] Potential slashing evasion during re-delegation

Severity Level	Low
Lines	https://github.com/cosmos/cosmos-sdk/tree/v0.46.7/x/staking/keeper/slash.go
Description	If a delegation contributed to byzantine behavior of a validator, and the validator has not yet been slashed, it may be possible for that delegation to evade a pending slashing penalty through re-delegation behavior.
Recommendation	It is recommended to add additional validation logic or upgrade the SDK version to v0.47.10 or higher.
Status	**Fixed.** The issue has been fixed in commit 02ba28d8c08a784891982c49aeec23ec10407a36 of the project. The current version of Cosmos-SDK being used is 0.47.10.

[Self Chain-07] The lockedAmount missing numeric check

Severity Level	Low
Lines	selfchain/x/migration/keeper/msg_server_migrate.go #L93-98

Description

In msg_server_migrate.go within selfchain/x/migration/keeper, the `AddBeneficiary` function performs a subtraction operation where `InstantlyReleasedAmount` is subtracted from `lockedAmount`. However, it lacks `pre-checking` to compare the values of `lockedAmount` and `InstantlyReleasedAmount`. Therefore, if `lockedAmount` is less `thanInstantlyReleasedAmoun`, an error occurs in the subtraction operation, causing this migration to fail.

```
    k.selfvestingKeeper.AddBeneficiary(ctx,
selfvestingTypes.AddBeneficiaryRequest{
        Beneficiary: msg.DestAddress,
        Cliff:       config.VestingCliff,
        Duration:    config.VestingDuration,
        Amount:      lockedAmount.Sub(types.GetInstantlyRelea
sedAmount()).String(),
    })
```

Recommendation

It is recommended to add a check that `lockedAmount` is greater than `InstantlyReleasedAmount`, and then perform the subtraction operation.

Status

Fixed.

```
    if migrationAmount.LTE(instantlyReleased) {
        instantlyReleasedCoins := sdk.NewCoins(sdk.NewCoin(
            types.DENOM,
            sdkmath.NewIntFromBigInt(migrationAmount.BigInt()),
        ))
        k.bankKeeper.SendCoinsFromModuleToAccount(ctx,
selfvestingTypes.ModuleName, destAddr, instantlyReleasedCoins)
    } else {
        instantlyReleasedCoins := sdk.NewCoins(sdk.NewCoin(
            types.DENOM,
            sdkmath.NewIntFromBigInt(instantlyReleased.BigInt())
        ))
        k.bankKeeper.SendCoinsFromModuleToAccount(ctx,
```

	```
	selfvestingTypes.ModuleName, destAddr, instantlyReleasedCoins)
	        k.selfvestingKeeper.AddBeneficiary(ctx,
	selfvestingTypes.AddBeneficiaryRequest{
	            Beneficiary: msg.DestAddress,
	            Cliff:       config.VestingCliff,
	            Duration:    config.VestingDuration,
	            Amount:      migrationAmount.Sub(instantlyReleased).Stri
	ng(),
	        })
	    }
	```

[Self Chain-08] Transaction failed due to wrong token type

Severity Level	Low
Lines	selfchain-migrator/src/blockchain/tx.rs #L38
Description	In tx.rs of selfchain-migrator/src/blockchain, when compute auth info by associating a fee, the token type used in auth.info is incorrectly written as uself, causing the gas balance to be judged to be insufficient when initiating a transaction. `let auth_info = signer_info.auth_info(Fee::from_amount_and_gas(Coin::new(5000, "uself")?, gas));`
Recommendation	It is recommended to Change uself to uslf.
Status	**Fixed.**

[Self Chain-09] Missing BlockedAddressed validation in vesting module

Severity Level	Low
Lines	https://github.com/cosmos/cosmos-sdk/tree/v0.46.7/x/auth/vesting/msg_server.go #L29-87

Description

There is a vulnerability in the "x/auth/vesting" module in the version of `Cosmos-SDK` used in this project that allows users to create regularly attributed accounts on blocked addresses, such as uninitialized module accounts.

If this is triggered, it may cause the chain to stop if `GetModuleAccount` in the module's `Begin`/`EndBlock` calls the relevant uninitialized account. This combination of uninitialized blocked module accounts is uncommon.

```
    from, err := sdk.AccAddressFromBech32(msg.FromAddress)
    if err != nil {
        return nil, err
    }
    to, err := sdk.AccAddressFromBech32(msg.ToAddress)
    if err != nil {
        return nil, err
    }
    if bk.BlockedAddr(to) {
        return nil, sdkerrors.Wrapf(sdkerrors.ErrUnauthorized, "%s is
not allowed to receive funds", msg.ToAddress)
    }
    if acc := ak.GetAccount(ctx, to); acc != nil {
        return nil, sdkerrors.Wrapf(sdkerrors.ErrInvalidRequest,
"account %s already exists", msg.ToAddress)
    }
    baseAccount := authtypes.NewBaseAccountWithAddress(to)
    baseAccount = ak.NewAccount(ctx,
baseAccount).(*authtypes.BaseAccount)
    baseVestingAccount := types.NewBaseVestingAccount(baseAccount,
msg.Amount.Sort(), msg.EndTime)
```

Recommendation

It is recommended to add additional validation was added to prevent creation of a periodic vesting account in this scenario or upgrade the SDK version to v0.47.9 or higher.

Status	**Fixed.** The issue has been fixed in commit 02ba28d8c08a784891982c49aeec23ec10407a36 of the project. The current version of `Cosmos-SDK` being used is 0.47.10.

[Self Chain-10] The target address in the DeadWallet contract is not verified

Severity Level	Info
Lines	token-migration/contracts/DeadWallet.sol
Description	The `destAddress` in the `DeadWallet` contract is not validated for basic correctness and formatting during migration. If a user mistakenly enters an address with an incorrect format, it can result in the user's funds being locked in the contract without the ability to mint them on Self Chain.

```
  function migrateFront(string memory destAddress)
whenOpen(FRONT_TOKEN) public {
    uint256 amount = front.balanceOf(msg.sender);
    require(amount >= MinMigrationAmount, "Insufficient FRONT
balance");
    front.safeTransferFrom(msg.sender, address(this), amount);
    emit NewMigration(msg.sender, FRONT_TOKEN, destAddress, amount);
  }

  function migrateHotcross(string memory destAddress)
whenOpen(HOTCROSS_TOKEN) public {
    uint256 amount = hotcross.balanceOf(msg.sender);
    require(amount >= MinMigrationAmount, "Insufficient HOTCROSS
balance");
    hotcross.safeTransferFrom(msg.sender, address(this), amount);
    emit NewMigration(msg.sender, HOTCROSS_TOKEN, destAddress,
amount);
  }
```

Recommendation	It is recommended to perform basic verification of the target address in the contract or user front-end.
Status	**Acknowledged.** The project team stated that it will perform verification on the front end and only allow wallet interaction.

[Self Chain-11] Redundant code

Severity Level	Info
Lines	selfchain/x/selfvesting/keeper/vesting_positions.go #L38-47
Description	In selfchain/x/selfvesting/keeper/vesting_positions.go, the purpose of the `RemoveVestingPositions` function is to delete vesting positions. However, in reality, this function is not used anywhere in the entire project and cannot be called externally. It is considered redundant code.
Recommendation	It is recommended to remove redundant code.
Status	**Fixed.**

```
// RemoveVestingPositions removes a vestingPositions from the store
func (k Keeper) RemoveVestingPositions(
    ctx sdk.Context,
    beneficiary string,
) {
    store := prefix.NewStore(ctx.KVStore(k.storeKey),
types.KeyPrefix(types.VestingPositionsKeyPrefix))
    store.Delete(types.VestingPositionsKey(
        beneficiary,
    ))
}
```

[Self Chain-12] Query function is missing parameter restrictions

Severity Level Info

Lines selfchain/x/migration/client/cli/query_migrator.go #L13-44

Description In selfchain/x/migration/client/cli/query_migrator.go, the implementation of the `CmdListMigrator` function lacks proper parameter configuration and validation, allowing arbitrary parameters to be added after the command.

```go
func CmdListMigrator() *cobra.Command {
    cmd := &cobra.Command{
        Use:   "list-migrator",
        Short: "list all migrator",
        RunE: func(cmd *cobra.Command, args []string) error {
            clientCtx := client.GetClientContextFromCmd(cmd)
            pageReq, err := client.ReadPageRequest(cmd.Flags())
            if err != nil {
                return err
            }
            queryClient := types.NewQueryClient(clientCtx)
            params := &types.QueryAllMigratorRequest{
                Pagination: pageReq,
            }
            res, err := queryClient.MigratorAll(context.Background(),
params)
            if err != nil {
                return err
            }

            return clientCtx.PrintProto(res)
        },
    }
    flags.AddPaginationFlagsToCmd(cmd, cmd.Use)
    flags.AddQueryFlagsToCmd(cmd)
    return cmd
}
```

Recommendation It is recommended to add `Args: cobra.NoArgs` to enforce parameter restrictions.

Status **Fixed.**

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

4.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

4.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

4.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.3 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com