Guvenkaya® The Bedrock of Security

Jump Defi

Backend And Frontend Security Assessment

Lead Security Engineer: Timur Guvenkaya
Date of Engagement: 10th December 2024 - 6th January 2025
Visit: www.guvenkaya.co

Contents

About Us

Guvenkaya is a security research firm specializing in Rust security, Web3 security of Rust-based protocols, and Web2 security. With our expertise, we provide both security auditing services and custom security solutions

About Jump Defi

Jump Defi is the only one-stop decentralized finance platform on NEAR Protocol. Jump DeFi lowers the barrier of entry to decentralized finance for users and developers.

Audit Results

Guvenkaya conducted a security assessment of the **Jump Defi backend** and **Jump Defi frontend** from 10th December 2024 to 6th January 2025. During this engagement, a total of **13 findings** were reported. 8 of the findings were critical, 1 high, 3 medium, and 1 low severity. All major issues were fixed by the Jump Defi team.

Project Scope

File name	Link
Indexer Helper (Excluding Backend Deployment Scripts)	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/indexer-helper
Cache	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/cache
Aggregator	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/aggregator
AMM Indexer	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/amm-indexer
Indexer	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/indexer
NEAR Service	https://github.com/BlockApex/jump-DeFi-backend/tree/e188e0a2d9ba419d9ace6eeaf3f9a5c8bc2f15ed/near_service
Web Package (Excluding tests and assets)	https://github.com/BlockApex/jump-web-dev/tree/0b6679ed376b430c5ae324e1e53c8e17bc6e0776/packages/web

Out of Scope

The audit will include reviewing the code for security vulnerabilities. The audit does not include a review of the tests and dependencies.

Timeline

Start of the audit	Draft report	Final report
10th December 2024	6th January 2025	18th January 2025

Methodology

- RESEARCH INTO PROJECT ARCHITECTURE
- PREPARING ATTACK VECTORS
- SETTING UP AN ENVIRONMENT
- MANUAL CODE REVIEW OF THE CODE
- ASSESSMENT OF RUST SECURITY ISSUES
- ASSESSMENT OF NEAR SECURITY ISSUES
- ASSESSMENT OF ARITHMETIC ISSUES
- BUSINESS LOGIC VULNERABILITY ASSESSMENT
- ONCHAIN TESTING USING NEAR WORKSPACES
- BEST PRACTICES AND CODE QUALITY
- CHECKING FOR CODE REFACTORING/SIMPLIFICATION POSSIBILITIES
- ARCHITECTURE IMPROVEMENT SUGGESTIONS
- PREPARING POCS AND/OR TESTS FOR EACH CRITICAL/HIGH/MEDIUM ISSUES

Severity Breakdown

01. Likelihood Ratings

Likely: The vulnerability is easily discoverable and not overly complex to exploit.
Possible: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Rare: The vulnerability is either very difcult to discover or complex to exploit, or both.
This matrix provides a nuanced view, taking into account both the ease of discovering a vulnerability and the complexity involved in exploiting it.

02. Impact

Severe: The vulnerability is easily discoverable and not overly complex to exploit.
Moderate: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Negligible: The vulnerability is either very difcult to discover or complex to exploit, or both.

03. Severity Ratings

Critical: Assigned to vulnerabilities with severe impact and a likely likelihood of exploitation.
High: For vulnerabilities with either severe impact but only a possible likelihood, or moderate impact with a likely likelihood.
Medium: Used for vulnerabilities with severe impact but a rare likelihood, moderate impact with a possible likelihood, or negligible impact with a likely likelihood.
Low: For vulnerabilities with moderate impact and rare likelihood, or negligible impact with a possible likelihood.
Informational: The lowest severity rating, typically for vulnerabilities with negligible impact and a rare likelihood of exploitation.

CRITICAL HIGH MEDIUM Low Informational

Likelihood Matrix:

Attack Complexity \ Discovery Ease	Obvious	Concealed	Hidden
Complex	Possible	Rare	Rare
Moderate	Likely	Possible	Rare
Straightforward	Likely	Possible	Possible

Likelihood/Impact Matrix:

Likelihood \ Impact	Severe	Moderate	Negligible
Likely	**CRITICAL**	**HIGH**	**MEDIUM**
Possible	**HIGH**	**MEDIUM**	Low
Rare	**MEDIUM**	Low	Informational

Findings Summary

01. Remediation Complexity: This measures how difcult it is to fx the vulnerability once it has been identifed.

Simple: Patches or fixes are readily available and easily implemented.
Moderate: Requires some time and resources to remediate, but well within the capabilities of most organizations.
Difficult: Remediation requires significant resources, specialized skills, or substantial changes to systems or architecture.

02. Status: This measures how difcult it is to fx the vulnerability once it has been identifed.

Not Fixed: Indicates that the vulnerability has been identifed but no remedial action has been taken yet. This status is crucial for newly discovered vulnerabilities or those awaiting prioritization.
Fixed: This status is applied when the vulnerability has been successfully remediated. It implies that appropriate measures (like patching, confguration changes, or architectural modifcations) have been implemented to resolve the issue.
Acknowledged: This status is used for vulnerabilities that have been recognized, but for various reasons (such as risk acceptance, cost, or other business decisions), have not been fxed. It indicates that the risk posed by the vulnerability is known and has been consciously accepted.

Finding	Impact	Likelihood	Severity	Remediation Complexity	Remediation Status
GUV-1: Indexer Crash Due to Invalid UTF-8 Character	Severe	Likely	CRITICAL	Simple	Fixed
GUV-2: Indexer Crash Due To Invalid Message Format	Severe	Likely	CRITICAL	Simple	Fixed
GUV-3: Aggregator Crash Due To Not Matching Pool	Severe	Likely	CRITICAL	Simple	Fixed
GUV-4: Indexer Crash Due to Invalid Argument Formats	Severe	Likely	CRITICAL	Simple	Fixed
GUV-5: Aggregator Crash Due To Integer Overflow	Severe	Likely	CRITICAL	Simple	Fixed
GUV-6: SQL Injection in User Wallet Addition	Severe	Likely	CRITICAL	Simple	Fixed
GUV-7: SQL Injection In Liquidation Result Adding	Severe	Likely	CRITICAL	Simple	Fixed
GUV-8: Indexer DoS Due To Unlimitted Messages	Moderate	Likely	CRITICAL	Simple	Fixed
GUV-9: Artificial Data Can Be Added Through AMM-Indexer	Moderate	Likely	HIGH	Simple	Fixed
GUV-10: Any Pool Can Be Marked As Blacklisted On Aggregator	Negligible	Likely	MEDIUM	Simple	Fixed
GUV-11: Missing Tests Across The System	Moderate	Possible	MEDIUM	Moderate	Acknowledged
GUV-12: Error-Prone Parsing of Liquidity Pools	Severe	Rare	MEDIUM	Moderate	Fixed
GUV-13: Usage of Panic Macro in Production Code	Moderate	Rare	Low	Simple	Fixed

Findings Details

GUV-1 Indexer Crash Due to Invalid UTF-8 Character - Critical

The Indexer uses **String::from_utf8** to convert base64-decoded argument bytes to strings. However, it **lacks proper error handling**and instead uses **unwrap()**, which **causes a panic** on None and Error results. The indexer also **fails to verify** whether transactions sent to valid addresses are successful.

indexer:handle_message:indexer/src/chains/near/stream.rs

```
let keywords = [
        self.config.ref_finance_contract_id.clone(),
        self.config.veax_contract_id.clone(),
        self.config.jump_contract_id.clone(),
    ];
    let contains_keyword = keywords.contains(&receiver_id);
    if contains_keyword {
        let r = receipt.receipt;
        if let ReceiptEnumView::Action { actions, .. } = &r {
            for action in actions {
                if let ActionView::FunctionCall { method_name, args, .. } = action {
                    let args_base64: String =
serde_json::from_value(serde_json::to_value(args).unwrap()).unwrap();
                    let decoded_args = STANDARD.decode(args_base64).unwrap();
                   let decoded_str = String::from_utf8(decoded_args).unwrap();
                    let mut decoded_json: Value = serde_json::from_str(&decoded_str).unwrap();}
```

The root causes:

- The indexer **does not verify** transaction success
- The indexer uses **unwrap()** instead of proper error handling, **causing panic** on None/Error results

This vulnerability enables:

- A malicious actor to **crash** the indexer with a single transaction

POC

- Compile the indexer in release mode and launch it
- Send a transaction to any valid contract address with an argument containing a non-UTF8 character. The transaction does not need to execute successfully
- Example:
  near contract call-function as-transaction ref-finance-101.testnet ft_transfer_call json-args "{"msg": "жжж"} prepaid-gas '100.0 Tgas' attached-deposit '0 NEAR' sign-as attacker.testnet network-config testnet sign-with-keychain send

Recommendation

- Verify that transactions are successful
- Implement proper error handling for all Option/Result values

Remediation - Fixed

The Jump Defi team has fixed the issue by verifying the success of the receipt and replaced unwrap() with proper error handling

GUV-2 Indexer Crash Due To Invalid Message Format - Critical

The Indexer attempts to parse the msg argument from JSON into an array for **ft_on_transfer** transactions sent to the **VEAX**address. The issue arises from using **unwrap()** after **.as_array()** without proper error handling. The indexer also fails to verify transaction success.

indexer:handle_message:indexer/src/chains/near/stream.rs

```
    if !self.config.exclude_veax && receiver_id == self.config.veax_contract_id
        && method_name == "ft_on_transfer"
        && !decoded_json["msg"].as_str().unwrap_or("").is_empty() {
            // msg contain stringify value, need to convert it to JSON
            let msg_json = serde_json::from_str(
                decoded_json["msg"].as_str().unwrap()
            ).unwrap();
            decoded_json["msgs"] = msg_json;

          for msg in decoded_json["msgs"].as_array().unwrap() {
                if let Some(obj) = msg.as_object() {
                    if obj.contains_key("OpenPosition") {
                        tracing::info!("stream::handle_message() Handling Veax OpenPosition Action");
```

The root causes:

- The indexer **does not verify** transaction success
- The indexer uses **unwrap()** instead of proper error handling, **causing panic** on None/Error results

This vulnerability enables:

- A malicious actor to **crash** the indexer with a single transaction

POC

- Compile the indexer in release mode and launch it
- Send a transaction to VEAX contract address with the msg argument containing a non-array value. The transaction does not need to execute successfully
- Example:
  near contract call-function as-transaction veax-dex15.testnett ft_on_transfer json-args{"msg":"hak"} prepaid-gas '100.0 Tgas' attached-deposit '0 NEAR' sign-as attacker.testnet network-config testnet sign-with-keychain send

Recommendation

- Verify that transactions are successful
- Implement proper error handling for all Option/Result values

Remediation - Fixed

The Jump Defi team has fixed the issue by verifying the success of the receipt and replaced unwrap() with proper error handling

GUV-3 Aggregator Crash Due To Not Matching Pool - Critical

The aggregator computes routes that are sent to the smart router contract. A mock router creation route ("/route/mock") is exposed in production without access control. Its **create_route** function attempts to fetch decimals for **token_in** and **token_out** by iterating over pools, but it uses **.expect()** instead of proper error handling.

aggregator:create_mock_route:aggregator/src/services/route_mock.rs

```
if let Some(pools) = mock_pools.clone() {
        token_in_decimals = pools
            .iter()
            .find_map(|pool| {
                if pool.token_in == token_in {
                    Some(pool.token_in_decimals)
                } else if pool.token_out == token_in {
                    Some(pool.token_out_decimals)
                } else {
                    None
                }
            })
           .expect("No matching pool found for token_in");

        token_out_decimals = pools
            .iter()
            .find_map(|pool| {
                if pool.token_in == token_out {
                    Some(pool.token_in_decimals)
                } else if pool.token_out == token_out {
                    Some(pool.token_out_decimals)
                } else {
                    None
                }
            })
           .expect("No matching pool found for token_out");
```

The root causes:

- The aggregator has exposed test route in production
- The aggregator uses .expect() instead of proper error handling, causing panics on None/Error results

This vulnerability enables:

- A malicious actor to crash the aggregator with a single request

POC

- Compile the aggregator in release mode and launch it
- Send the request to the /route/mock with token_in and/or token_out which do not match any of the pools:

Example request

```
curl -X POST 'http://127.0.0.1:8000/route/mock' -H 'Content-Type: application/json'          -d
'{
            "token_in": "token121.near",
            "token_out": "token212.near",
            "amount": 1000.0, "slippage": "0.5",
            "providers": null, "pools": [{
               "kind": "STABLE_SWAP", "liquidity_provider": "RefFinance",
               "amp": 100, "key": "pool_key_1",
               "pool_id": 1, "token_in": "token1.near",
               "token_out": "token2.near", "reserves_in": "1000000",
               "reserves_out": "1000000", "fees": null,
               "fee_divisor": null, "liquidities": ["1000000", "1000000"],
               "prices": null, "spot_prices": null,
               "symbols": ["TOKEN1", "TOKEN2"], "token_in_decimals": 18,
               "token_out_decimals": 18, "amounts": ["1000000", "1000000"],
               "token_account_ids": ["token1.near", "token2.near"],
               "decimals": [18, 18], "swap_path": {
                  "token_in": "token1.near", "token_out": "token2.near",
                  "amount_in": "1000", "amount_out": "995",
                  "internal_swap": null, "actual_order_book": null,
                  "calculated_order_book": null, "order_type": null,
                  "order_quantity": null,"slippage_tolerance_bp": null
               }
            }]
         }'
```

Recommendation

- Remove the mock route from production code by utilizing test config
- Implement proper error handling for all Option/Result values

Remediation - Fixed

The Jump Defi team has fixed the issue by removing the mock route.

GUV-4 Indexer Crash Due to Invalid Argument Formats - Critical

For each supported method, the Indexer attempts to parse arguments from JSON into their respective types. The issue stems from using **unwrap()** without proper error handling. Additionally, the indexer fails to verify transaction success.

indexer:handle_message:indexer/src/chains/near/stream.rs

```
if ["ft_transfer_call"].contains(&method_name.as_str()) {

      if !decoded_json["msg"].as_str().unwrap_or("").is_empty() {
        let msg_json = serde_json::from_str(
          decoded_json["msg"].as_str().unwrap()
        ).unwrap();

        decoded_json["msg"] = msg_json;

       let ft_transfer_call: FtTransferCall =
from_value(decoded_json.clone()).unwrap();
        actions = ft_transfer_call.msg.actions;
      }
    }
    ...
    if method_name == "add_liquidity" {

    let add_liquidity: AddLiquidity =
      from_value(decoded_json.clone()).unwrap();...}
    ...
    if method_name == "remove_liquidity" {

    let remove_liquidity: RemoveLiquidity =
      from_value(decoded_json.clone()).unwrap();...}
```

indexer:handle_message:indexer/src/chains/near/stream.rs

```
    if method_name == "add_stable_liquidity" {
                tracing::info!(
               "stream::handle_message() Handling {} method",
                    method_name
                );

               let add_stable_liquidity: AddStableLiquidity =
                from_value(decoded_json.clone()).unwrap();
        ...
        for msg in decoded_json["msgs"].as_array().unwrap() {
            if let Some(obj) = msg.as_object() {
                if obj.contains_key("OpenPosition") {
                    tracing::info!(
                        "stream::handle_message() Handling Veax OpenPosition Action"
                    );
               let open_position: OpenPosition =
                 from_value(msg["OpenPosition"].clone()).unwrap();
```

The affected methods include:

- ft_transfer_call ⇒ FtTransferCall type
- add_liquidity ⇒ AddLiquidity
- remove_liquidity ⇒ RemoveLiquidity
- add_stable_liquidity ⇒ AddStableLiquidity
- ft_on_transfer (VEAX) ⇒ OpenPosition

The root causes:

- The indexer does not verify transaction success
- The indexer uses unwrap() instead of proper error handling

POC

- Compile the indexer in release mode and launch it
- Send a transaction to any valid contract address with supported method and an argument which does not match the type expected by the method
- Example:
  near contract call-function as-transaction ref-finance-101.testnet remove_liquidity json-args '"something":33' prepaid-gas '100.0 Tgas' attached-deposit '0 NEAR' sign-as attacker.testnet network-config testnet sign-with-keychain send

Recommendation

- Verify that transactions are successful
- Implement proper error handling for all Option/Result values

Remediation - Fixed

The Jump Defi team has fixed the issue by verifying the success of the receipt and replaced unwrap() with proper error handling

GUV-5 Aggregator Crash Due To Integer Overflow - Critical

The Aggregator allows users to calculate routes by accepting various arguments, including the **amount** parameter. The core issue lies in the use of **unchecked mathematics** operations throughout the codebase.

aggregator:get_amounts_distribution:aggregator/src/services/route.rs

```
    fn get_amounts_distribution(amount: u128, distribution_percentage: u128) -> (Vec<u128>,
    Vec<u128>) {
        tracing::info!("router::get_amounts_distribution() getting amounts distribution...");

        let mut amounts: Vec<u128> = Vec::new();
        let mut percents: Vec<u128> = Vec::new();

        let iterations = (100u128 / distribution_percentage) as usize;

        for i in 1..=iterations {
            let percent = distribution_percentage * i as u128;
          let amount = amount * percent / 100u128;

            percents.push(percent);
            amounts.push(amount);
        }

        (amounts, percents)
      }
```

The root cause:

- The aggregator uses unchecked maths throughout the code

POC

- Compile the aggregator in release mode and launch it
- Send the request to the /route with amount which equals to the u128::MAX

Example request

```
curl -X POST http://127.0.0.1:8000/route -H "Content-Type: application/json" -d           '{
          "token_out": "usdt.tether-token.near",
          "token_in": "wrap.near",
          "amount": 340282366920938463463374607431768211455,
          "slippage": "0.5",
          "providers": ["Jump", "RefFinance"]
        }'
```

Recommendation

Utilize checked maths throughout the aggregator

Remediation - Fixed

The Jump Defi team has fixed the issue by utilizing the checked maths.

GUV-6 SQL Injection in User Wallet Addition - Critical

The indexer helper exposes an endpoint for adding user wallets to the database via **/add-user-wallet**. The endpoint lacks verification of provided arguments before constructing SQL queries. Both **account_id** and wallet_address parameters are directly concatenated into the query string.

indexer_helper:add_user_wallet_info:indexer-helper/db_provider.py

```
query_sql = "select id from t_user_wallet_info where account_id = '%s' and
wallet_address = '%s'" % (account_id, wallet_address)
    sql = "insert into t_user_wallet_info(account_id, wallet_address, `created_time`,
`updated_time`) " \
          "values('%s','','b','c'%s', now(), now())" % (account_id, wallet_address)
     cursor = db_conn.cursor()
     try:
       cursor.execute(query_sql)
       row = cursor.fetchone()

       if row is None:
         cursor.execute(sql)
         db_conn.commit()
     except Exception as e:
       db_conn.rollback()
       print("insert t_user_wallet_info to db error:", e)
       raise e
```

The root causes:

- Missing access control on /add-user-wallet endpoint
- Missing validation of user-provided arguments

This vulnerability enables a malicious actor to perform SQL injection attacks that can:

- Read unauthorized data from the database
- Insert malicious data into the database
- Cause denial of service on the database
- Potentially execute arbitrary code remotely

POC

- Launch indexer helper
- Send request to /add-wallet-connection with Sleep command

Example request

```
curl -X POST "http://localhost:9000/add-user-wallet" -H "Content-Type: application/json" -
d '{
                    "account_id": "test_account3333",
                    "wallet_address": "' UNION SELECT 1 AND SLEEP(20) -- "
                }'
```

Recommendation

- Add access control to the add-user-wallet endpoint
- Provide named parameters in .execute to prevent SQL injection

Remediation - Fixed

The Jump Defi team has fixed the issue by removing the add-user-wallet endpoint.

GUV-7 SQL Injection In Liquidation Result Adding - Critical

The indexer helper exposes an endpoint for adding liquidation result to the database via **/add-liquidation-result**. The endpoint lacks verification of provided arguments before constructing SQL queries. Both key and values parameters are directly concatenated into the query string.

indexer_helper:add_liquidation_result:indexer-helper/db_provider.py

```
    sql = "insert into liquidation_result_info(`key`, `values`, `timestamp`, `created_time`,
    `updated_time`) " \
          "values('%s','%s',%s, now(), now())" % (key, values, now_time)
        cursor = db_conn.cursor(cursor=pymysql.cursors.DictCursor)
        try:
          cursor.execute(sql)
           db_conn.commit()
```

The root causes:

- Missing access control on /add-liquidation-result endpoint
- Missing validation of user-provided arguments

This vulnerability enables a malicious actor to perform SQL injection attacks that can:

- Read unauthorized data from the database
- Insert malicious data into the database
- Cause denial of service on the database
- Potentially execute arbitrary code remotely

POC

- Launch indexer helper
- Send request to /add-liquidation-result with Sleep command

Example request

```
curl -X POST "http://localhost:9000/add-liquidation-result" -H "Content-Type:
application/json" -d '{
            "key": "pwned','pwned632622',(SELECT SLEEP(50)),'2023-01-01','2023-01-01')-- ",
            "values": "1"
        }'
```

Recommendation

- Add access control to the add-liquidation-result endpoint
- Provide named parameters in .execute to prevent SQL injection

Remediation - Fixed

The Jump Defi team has fixed the issue by removing the add-liquidation-result endpoint.

GUV-8 Indexer Denial Of Service Due To Exceesive Number Of Messages - Critical

The Indexer iterates over a messages array to process individual messages. Since it verifies neither the number of messages nor the success of the receipt, an attacker can force the indexer to process a call containing an excessive number of messages. This prevents the indexer from processing any other blocks until it completes the current messages, resulting in a denial of service.

indexer:handle_message:indexer/src/chains/near/stream.rs

```
for msg in decoded_json["msgs"].as_array().unwrap() {
            if let Some(obj) = msg.as_object() {
                if obj.contains_key("OpenPosition") {
                    tracing::info!("stream::handle_message() Handling Veax OpenPosition Action");
```

The root cause:

- The indexer **does not verify** transaction success

This vulnerability enables:

- A malicious actor to cause **denial of service** of the indexer

POC

- Compile the indexer in release mode and launch it
- Send a transaction to VEAX contract address with the msg argument containing a large number of entries

Example exploit code:

exploit

```
import json

      def generate_open_position(index):
         return {
            "OpenPosition": {
               "fee_rate": 30,
               "position": {
                  "amount_ranges": [{
                     "max": "1000000000",
                     "min": "100000000"
                  }],
                  "ticks_range": [100 + index, 200 + index]
               },
               "tokens": [f"token{index}a.testnet", f"token{index}b.testnet"]
            }
         }

      # Generate array with 100 messages
      messages = [generate_open_position(i) for i in range(100)]

      # Create the full payload
      payload = {
         "sender_id": "malicious_acc.testnet",
         "amount": "1000000000",
         "msg": json.dumps(messages)
      }
```

exploit

```
    # Write the command to a file
            command = f"""near contract call-function as-transaction veax-dex15.testnet
    ft_on_transfer json-args '{json.dumps(payload)}' prepaid-gas '100.0 Tgas' attached-deposit
    '0 NEAR' sign-as malicious_acc.testnet network-config testnet sign-with-keychain send"""

            with open("run_attack.sh", "w") as f:
               f.write(command)

            print("Command written to run_attack.sh")

            # Also print just the messages array for inspection
            print("
    Messages array (first 2 for verification):")
            print(json.dumps(messages[:2], indent=2))
```

Recommendation

Verify that transactions are successful

Remediation - Fixed

The Jump Defi team has fixed the issue by verifying the success of the receipt.

GUV-9 Artificial Data Can Be Added Through AMM-Indexer - High

The AMM indexer saves transaction data to the database based on the method called. Due to **missing transaction validation**, malicious users can **insert artificial or incorrect data** into the database.

Validation Issues per supported method:

- ft_transfer_call | add_liquidity | remove_liquidity ⇒ No validation of transaction success, allowing anyone to update pool data and add transactions to the database
- swap ⇒ No validation of transaction success, allowing anyone to add transactions to the database
- add_simple_pool ⇒ Has transaction success validation, but the receiver ID check can be bypassed

amm_indexer:handle_streamer_message:amm-indexer/src/main.rs

```
if receiver_id.contains("jump_amm.testnet") {
    let timestamp = streamer_message.block.header.timestamp_nanosec;
    let signer_id = receipt.predecessor_id.to_string();
    let receipt_id = receipt.receipt_id.to_string();
```

An attacker can:

- Create an account such as jump_amm.testnet.attackerjump.testnet, which will bypass this check
- Deploy smart contract with the function add_simple_pool which returns success
- Call jump_amm.testnet.attackerjump.testnet with any add_simple_pool arguments to save the data to the indexer database

This vulnerability enables:

- Bloating the database to make it costly to run and query
- Causing system components to operate on invalid data

Recommendation

Transaction success must be verified, and exact equality should be used instead of contains() when matching supported contracts

Remediation - Fixed

The Jump Defi team has fixed the issue by verifying the success of the receipt and replaced the contains() with exact equality.

GUV-10 Any Pool Can Be Marked As Blacklisted On Aggregator - Medium

The Aggregator's **/pool** route allows pools to be blacklisted without any access control mechanism.

aggregator:update_pool:aggregator/src/main.rs

```
#[put("/pool")]
    async fn update_pool(payload: web::Json<BlacklistPool>) -> impl Responder {
        let BlacklistPool {
            pool_id,
            liquidity_provider,
            token_in,
            token_out,
            blacklisted,
        } = payload.into_inner();
        tracing::info!(
            "main::blacklist_pool() - Updating pool with pool_id: {}, blacklisted: {}",
            pool_id,
            blacklisted
        );

        let key = format!(
            "pool:{}:{}:{}:{}",
            liquidity_provider, pool_id, token_in, token_out
        );

        match pool::update_pool(&key, blacklisted)
```

Recommendation

Implement access control for this route

Remediation - Fixed

The Jump Defi team has fixed the issue by removing the pool endpoint.

GUV-11 Missing Tests Across The System - Medium

The system components **lack test coverage**. Given the numerous cases and complex calculations performed by the indexers and aggregator, comprehensive test coverage is essential.

Recommendation

Write both unit and integration tests for indexer, amm-indexer, and aggregator

Remediation - Acknowledged

The Jump Defi team has acknowledged the issue and will fix it later.

GUV-12 Error-Prone Parsing of Liquidity Pools - Medium

The **parse_liquidity_pools** function relies on array indexes to extract data from bulk_data. This approach is error-prone as changes to data format or missing fields will cause crashes. The function's validation is also incorrect — it checks **if bulk_data.len() >= 8**, but accesses index 43.

cache:parse_liquidity_pools:cache/src/services/parser.rs

```
if let Value::Bulk(bulk_data) = value {
        // Extract required data from bulk data
        if bulk_data.len() >= 8 {
          let pool_id = get_value_as_string(&bulk_data[1]);
          let liquidity_provider = get_value_as_string(&bulk_data[3]);
            let token_in: String = get_value_as_string(&bulk_data[5]);
           ...
          let chain_id = get_value_as_string(&bulk_data[43]);
```

Recommendation

- Do not rely on indexes to fetch data from Redis
- Implement a proper deserializer using FromRedisValue trait for the expected type

Remediation - Fixed

The Jump Defi team has fixed the issue by rewriting the logic to parse the bulk data without relying on indexes.

GUV-13 Usage of Panic Macro in Production Code - Low

The **divide_and_floor** function uses the **panic!** macro when encountering a zero divisor. Although this function is currently unused in the main codebase, it should be removed to prevent potential crashes if implemented in the future.

aggregator:divide_and_floor:aggregator/src/services/pool.rs

```
fn divide_and_floor(&self, dividend: BigUint, divisor: BigUint) -> BigUint {
        if divisor.is_zero() {
          panic!("Division by zero");
        }
          dividend / divisor // In BigUint, division is inherently floor division
      }
```

Recommendation

Remove divide_and_floor function

Remediation - Fixed

The Jump Defi team has fixed the issue by removing the divide_and_floor function.