Guvenkaya® The Bedrock of Security

Cleopetra

Solana Trading Bot Security Review

Lead Security Engineer: Timur Guvenkaya
Date of Engagement: 13th May 2025 - 26th May 2025
Visit: www.guvenkaya.co

Contents

About Us

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

About Cleopetra

Cleopetra is a seamless platform for liquidity provisioning on Solana DEXes, directly from Telegram. Enter a token, pick an LP strategy of your choice and create the position — the agent finds optimal pools and auto-rebalance positions when needed to maximise fees and reduce impermanent loss.

Audit Results

Guvenkaya conducted a security assessment of the Cleopetra **Solana Trading Bot** from 13th May 2025 to 26th May 2025. During this engagement, a total of **12 findings** were reported. 6 of the findings were medium and the remaining were either low or informational severity. All major issues are either fixed or acknowledged by the Cleopetra team.

Project Scope

Files	Link
Bot	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/bot
Cron	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/cron
DB	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/db
Helpers	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/helpers
Config	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/config.ts
Constants	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/constants.ts
Index	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/index.ts
Providers	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/providers.ts
Types	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/types.ts
Utils	https://github.com/cleopetrafun/cleo/tree/f22cf033a335d2717245fbc8ba7cd327a0215eb2/src/utils.ts

Out of Scope

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

Timeline

Start of the audit	Draft report	Final report
13th May 2025	26th May 2025	21st June 2025

Methodology

- RESEARCH INTO PROJECT ARCHITECTURE
- PREPARING ATTACK VECTORS
- SETTING UP AN ENVIRONMENT
- MANUAL CODE REVIEW OF THE CODE
- ASSESSMENT OF JS/TS SECURITY ISSUES
- ASSESSMENT OF SOLANA SPECIFIC SECURITY ISSUES
- ASSESSMENT OF ARITHMETIC ISSUES
- BUSINESS LOGIC VULNERABILITY ASSESSMENT
- BEST PRACTICES AND CODE QUALITY
- CHECKING FOR CODE REFACTORING/SIMPLIFICATION POSSIBILITIES
- ARCHITECTURE IMPROVEMENT SUGGESTIONS
- PREPARING POCS AND/OR TESTS FOR RELEVANT 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: Exploitation could result in critical loss or compromise, such as full system control, substantial financial loss, or severe reputational damage.
Moderate: Exploitation may lead to limited data loss, partial compromise, moderate financial impact, or noticeable degradation of services.
Negligible: Exploitation has minimal impact, such as minor data exposure without significant consequences or slight inconvenience without substantial disruption.

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.

Scheduled: This status indicates that the vulnerability has been acknowledged and a plan is in place to fix it in the future. It signifies that while remediation hasn't yet occurred, the issue has been prioritized and is part of the planned development roadmap.

Finding	Impact	Likelihood	Severity	Remediation Complexity	Remediation Status
GUV-1: Incorrect Use of Async in Reward Distribution	Moderate	Possible	**MEDIUM**	Simple	Fixed
GUV-2: Possible Loss of Funds Through Invalid State Update	Moderate	Possible	**MEDIUM**	Simple	Fixed
GUV-3: Insecure Private Key Export Flow	Severe	Rare	**MEDIUM**	Simple	Fixed
GUV-4: No Encryption-at-rest for Sensitive User Data	Severe	Rare	**MEDIUM**	Simple	Scheduled
GUV-5: Database Tampering Risk	Severe	Rare	**MEDIUM**	Simple	Fixed
GUV-6: Potential Private Key Leakage in Error Logging	Moderate	Possible	**MEDIUM**	Simple	Fixed
GUV-7: Incorrect Units for changing priority fees	Negligible	Likely	Low	Simple	Fixed
GUV-8: Potential Race Condition in Mass Operations	Moderate	Rare	Low	Simple	Acknowledged
GUV-9: Insufficient Onchain Transaction Confirmation	Moderate	Rare	Low	Simple	Acknowledged
GUV-10: Preventing Possible Insecure Direct Object Reference	Negligible	Rare	Informational	Simple	Fixed
GUV-11: Lack Of Health/Execution Logging and Error Recovery	Negligible	Rare	Informational	Simple	Fixed
GUV-12: Inadequate Confirmation for Token Recipient and Amount	Negligible	Rare	Informational	Simple	Fixed

Findings Details

GUV-1 Incorrect Use of Async in Reward Distribution - Medium

The distributeRewardsJob uses *Map.forEach(async () =>)* to group referrers and send notifications. But forEach doesn't wait for async code to finish. This means important steps like building the reward batches and notifying users might still be running while the function moves on to sending transactions and updating state.

This can cause multiple issues, such as:

- Some users might not get the rewards they earned.
- Shared data might be changed at the same time from different places, leading to bugs or incorrect results.
- The link between who referred whom and what rewards they got might break.
- Rewards could be lost, database records might be wrong, and users could lose trust in the system.

POC: GUV-1

Recommendation

Replace all map.forEach(async ... usage for asynchronous workflows with sequential for...of loops or bulk Promise.all() approach. Carefully construct batches first, then proceed to transactional stages.

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-2 Possible Loss of Funds Through Invalid State Update - Informational

Reward distribution job attempt to sign and send multiple transactions in a loop, pushing each signature to an array. However, there is no granular verification or handling for failed transactions per user/referrer. If **sendAndConfirmTransaction** fails for a batch or individual reward payout, downstream logic still marks rewards as processed and notifies users, even if the transaction failed on-chain.

POC: **GUV-2**

Recommendation

Track transaction success/failure and only mark as processed those fees which actually receive confirmed on-chain rewards and for failed transactions, implement retry logic or mark for audit/reprocessing.

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-3 Insecure Private Key Export Flow - Medium

The exportPrivateKey command allows users to export their (Privy-based) private keys directly through a Telegram message with:

/src/bot/composers/wallet/exportPrivateKey.ts

```
    await ctx.reply('✅ Successfully exported private key -

        '${privateKey}'

         Import private key into wallets like Phantom (tap to copy) and delete this message
    once you are done.', { parse_mode: "Markdown" });
```

Exported keys are sent as plaintext in a chat message, putting them at risk from Telegram/Telegram client interception and device compromise. The export implementation in /src/helpers/privy.ts is invoked directly on request, bypassing secure delivery or split-secret protections.

Recommendation

it is imperative to implement a mandatory secondary authentication system PIN OR a user-defined passphrase before allowing any sensitive operations such as private key export.

During the initial wallet setup, require users to create a secure PIN OR passphrase and enforce a multi-step confirmation flow:

- Prompt users to re-enter their PIN/passphrase before proceeding.
- Display a strong security warning with a deliberate confirmation step (e.g., type "CONFIRM").

The Cleopetra team has fixed this issue in this PR: #99

GUV-4 No Encryption-at-rest for Sensitive User Data - Medium

The PostgreSQL schema for users stores **privyWalletAddress**,**privyWalletId**, and possibly private key export activity (didExportPvtKey) in plaintext (see /src/db/schema.ts). There is no evidence of encryption-at-rest, field-level encryption for sensitive columns, or defense-in-depth for backend storage. While wallet private keys are not directly stored, IDs are. If the database is compromised (insider threat, credential leak, SQLi), attackers may tamper with the wallet ids and drain user funds.

Recommendation

- Encrypt sensitive user columns at rest with application-level encryption (encryption/decryption key separate from DB, e.g. KMS-backed).
- Ensure database is encrypted on-disk using PostgreSQL's built-in encryption or cloud-provided at-rest encryption.
- Only expose Privy wallet IDs for short-lived server processes; avoid long-term storage unless required.

Remediation - Scheduled

The Cleopetra Team has scheduled the fix.

GUV-5 Database Tampering Risk - Medium

The current implementation of the bot stores both Privy wallet IDs and addresses in the database. This creates a security risk where an attacker who gains database access could modify wallet addresses while keeping the original wallet IDs. This would allow them to execute transactions that are signed by legitimate wallets but paid from attacker-controlled addresses for example when user calls **closePosition** function that would affect all position management operations including closing positions and can exploited by anyone with database write access.

Recommendation

- Encrypt sensitive user columns at rest with application-level encryption (encryption/decryption key separate from DB, e.g. KMS-backed).
- Ensure database is encrypted on-disk using PostgreSQL's built-in encryption or cloud-provided at-rest encryption.
- Only expose Privy wallet IDs for short-lived server processes; avoid long-term storage unless required.

DatabaseTrigger

```
CREATE OR REPLACE FUNCTION prevent_privy_wallet_id_update()
      RETURNS TRIGGER AS $$
      BEGIN
        IF NEW.privy_wallet_id IS DISTINCT FROM OLD.privy_wallet_id THEN
          RAISE EXCEPTION 'privy_wallet_id is immutable and cannot be changed';
        END IF;
        RETURN NEW;
      END;
      $$ LANGUAGE plpgsql;
```

Remediation - Fixed

The Cleopetra Team has fixed the by adding a trigger which restricts update for privy_wallet_id column and restrict regular db users to alter/drop that function.

GUV-6 Potential Private Key Leakage in Error Logging - Medium

The private key export functionality contains a security issue in the **exportPrivateKey** function where sensitive information about private key operations is might be logged in error messages and this concerning because the error message from the underlying operation could potentially contain sensitive information about the private key export process

exportPrivateKey

```
    console.log(
        'an error occurred while exporting pvt key for ${walletId} wallet - ${e.message}'
      );
      throw err;
```

Recommendation

Remove or sanitize the error logging to avoid exposing sensitive information in private key export functionality OR use a dedicated error logging service that can automatically sanitize sensitive information.

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-7 Incorrect Units for changing priority fees - Low

The **changePriorityFees** accepts a SOL amount typed by the user, parses it with **parseFloat**, stores the raw value directly in **users.computeUnitsPrice**, and later re-uses it as lamports/μ-lamports inside transaction-building logic:

```
/src/bot/composers/settings/changePriorityFees.ts

    const priorityFees = parseFloat(ctx.message.text);
        await db.update(usersTable).set({ computeUnitsPrice: priorityFees })
```

The value is then consumed as:

```
/src/helpers/solana.ts

    microLamports: cuPrice * 0.3,    // expects μ-lamports per CU
        lamports: cuPrice * 0.7        // Jito tip transfer (expects lamports)
```

since cuPrice is supposed to represent micro-lamports / CU (i.e. 1e-6 SOL units). The same value is blindly multiplied by 0.7 and used as a direct lamport transfer to the Jito tip account. For the default **MEDIUM_PRIORITY** the bot tips 0.07 SOL per transaction (100 000 lamports × 0.7), more than 700× the intended 0.1 lamport/CU.

Recommendation

Store computeUnitsPrice exclusively in μ-lamports (integer). Convert SOL → μ-lamports when accepting human input: *Math.floor(inputSol * LAMPORTS_PER_SOL / 1_000_000)*.

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-8 Potential Race Condition in Mass Operations - Low

When performing mass-operations ("close all positions", "claim all fees"), the bot executes actions in parallel (Promise.all) on a mutable ctx.session.positions array. If array/context is modified mid-operation (by another event, user spam, concurrent callback, or user race), there could be:

- Operations on non-existent/unstable state (double close, duplicate claim, non-existent index).
- State desync leading to incorrect user feedback, failed tx, or even race-triggered wallet logic bugs.

Example - Close All Positions (portfolio:close_all:yes callback):

CloseAllPositions

```
const promises = ctx.session.positions.map((position, i) => {
       return new Promise<void>(async (resolve, _) => {
         // ... position closing logic ...
       });
     });
     await Promise.all(promises);
```

There's no locking mechanism to prevent concurrent mass operations the array could be modified by other operations while these mass operations are running

Recommendation

create a lock mechanism to prevent concurrent mass operations, below is example fix for Close All Positions it's recommended to do the same for all Mass Operations like:

- Claim All Fees
- Mass DM

Remediation - Acknowledged

The Cleopetra Team has acknowledged the issue.

GUV-9 Insufficient Onchain Transaction Confirmation - Low

Transactions are sent and confirmation is checked (confirmTransaction, parseTxn) for "confirmed" status, but in Solana, "confirmed" is not durable finality ("finalized" is safest). The bot proceeds to update database state (positions, fees, etc.) and notifies users immediately on "confirmed", which may be reverted or dropped in rare Solana edge cases (such as forks or cluster issues). There is no check for "finalized" status, and database updates are not re-synced if transactions revert later.

Example

```
const metadata = await connection.getParsedTransaction(sig, {
          commitment: "confirmed",
          maxSupportedTransactionVersion: 0,
         });
```

Recommendation

Use "finalized" commitment instead of "confirmed".

Remediation - Acknowledged

The Cleopetra Team has acknowledged the issue.

GUV-10 Preventing Possible Insecure Direct Object Reference - Informational

While the current implementation in our Telegram bot doesn't present an immediate IDOR vulnerability (as parameters are controlled through Telegram's interface), it's a security best practice to implement defense in depth by enforcing strict ownership verification for all database operations. This approach ensures that even if the application's interface changes or new access points are added in the future, the underlying data access layer maintains proper security controls.

Examples from the current implementation missing explicit ownership verification:

Example

```
const positionInfo = await db.query.positionsTable.findFirst({
        where: eq(positionsTable.address, positionAddress)
      });
```

Example

```
await db
        .update(positionsTable)
        .set({
          rebalancingEnabled: !positionInfo.rebalancingEnabled
        })
        .where(eq(positionsTable.address, ctx.session.positionAddress));
```

Recommendation

This how the recommended implementation would look like

PotentialSolution

```
const positionInfo = await db.query.positionsTable.findFirst({
        where: and(
          eq(positionsTable.address, positionAddress),
          eq(positionsTable.owner, ctx.session.wallet)
        )
      });
```

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-11 Lack Of Health/Execution Logging and Error Recovery - Informational

In the cronjobs Logging is limited to console.log statements, with some errors only sent as Discord messages. There is no structured health monitoring, persistent error reporting/retries, or success/failure audit logs outside of ephemeral console/Discord. Issues may therefore go undetected or unresolved:

- Missed/failed cron runs.
- Partial batch failures.
- Recurring errors causing fund or data integrity issues

Recommendation

- Integrate **DataDog** or **Sentry** for comprehensive monitoring using the typescript SDK.
- Add structured logging with consistent format including timestamps.
- Implement retry mechanisms with exponential backoff Use existing **retryFuncWrapper**

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #93

GUV-12 Inadequate Confirmation for Token Recipient and Amount - Informational

Before final transmission of funds (SOL or SPL tokens) to the user-provided vault address: The bot does not explicitly show a "confirm transfer" summary of critical details (recipient, token/mint, amount/decimal) once more before sending. Users may make mistakes due to typos, malware-autofill, or clipboard hijacking, leading to fund loss.

Recommendation

- Insert an explicit human-confirmation step after collecting all transfer details (recipient, token, amount), before executing the blockchain transaction. Require the user to click a "confirm" button, displaying full destination address and parsed amount.
- Provide a last-moment "cancel" option in the confirmation step.

Remediation - Fixed

The Cleopetra team has fixed this issue in this PR: #96