Cyberscope

Audit Report

Kommunitas

July 2024

Repository	https://github.com/Kommunitas-net/telegram-bot
Commit	c1a95521bad5fbd67d9e56b171e0778f6312e2c2
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/Kommunitas-net/telegram-bot
Commit	c1a95521bad5fbd67d9e56b171e0778f6312e2c2
Assessment Scope	Telegram Bot

Audit Updates

Initial Audit	03 Jul 2024

Overview

Cyberscope has conducted a comprehensive penetration test on the Kommunitas' Telegram Bot. This report focuses on evaluating the security and performance aspects of the bot. The assessment encompasses various facets of the application, including but not limited to authentication and authorization mechanisms, data handling and storage practices, network security measures, and response to high traffic volumes.

The expansion of blockchain technology has introduced a myriad of innovative applications, each with its own unique security challenges. Kommunitas, as a prime example within the realm of digital currency ecosystems, ensures robust protection of user data and system integrity.

Assessment Scope

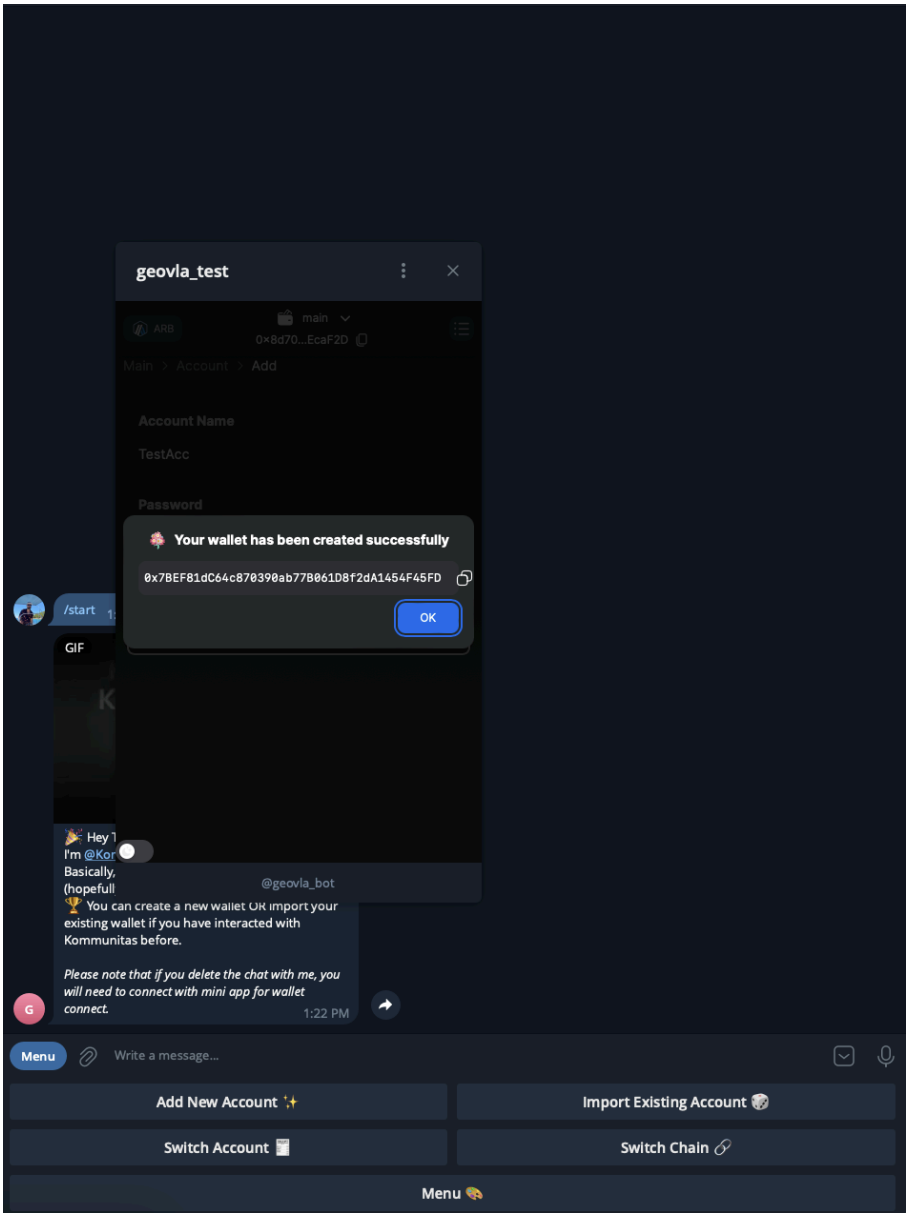
The scope of this assessment extends to identifying vulnerabilities and weaknesses in the application's architecture and functionality, with the aim of providing actionable recommendations to enhance its security posture. The evaluation focused specifically on the telegram bot of the ecosystem. The report aims to offer a comprehensive understanding of the application's strengths and areas for improvement, facilitating informed decision-making to mitigate risks, fortify against potential cyber threats, and bolster overall security resilience.

Bot Testing Preview

Our team has successfully tested the Telegram bot through the Telegram app, ensuring its functionality and reliability. The bot passed all the conducted tests, demonstrating its capability to handle various operations seamlessly. The tests focused on key features that users interact with frequently, and the bot performed exceptionally well in each scenario.

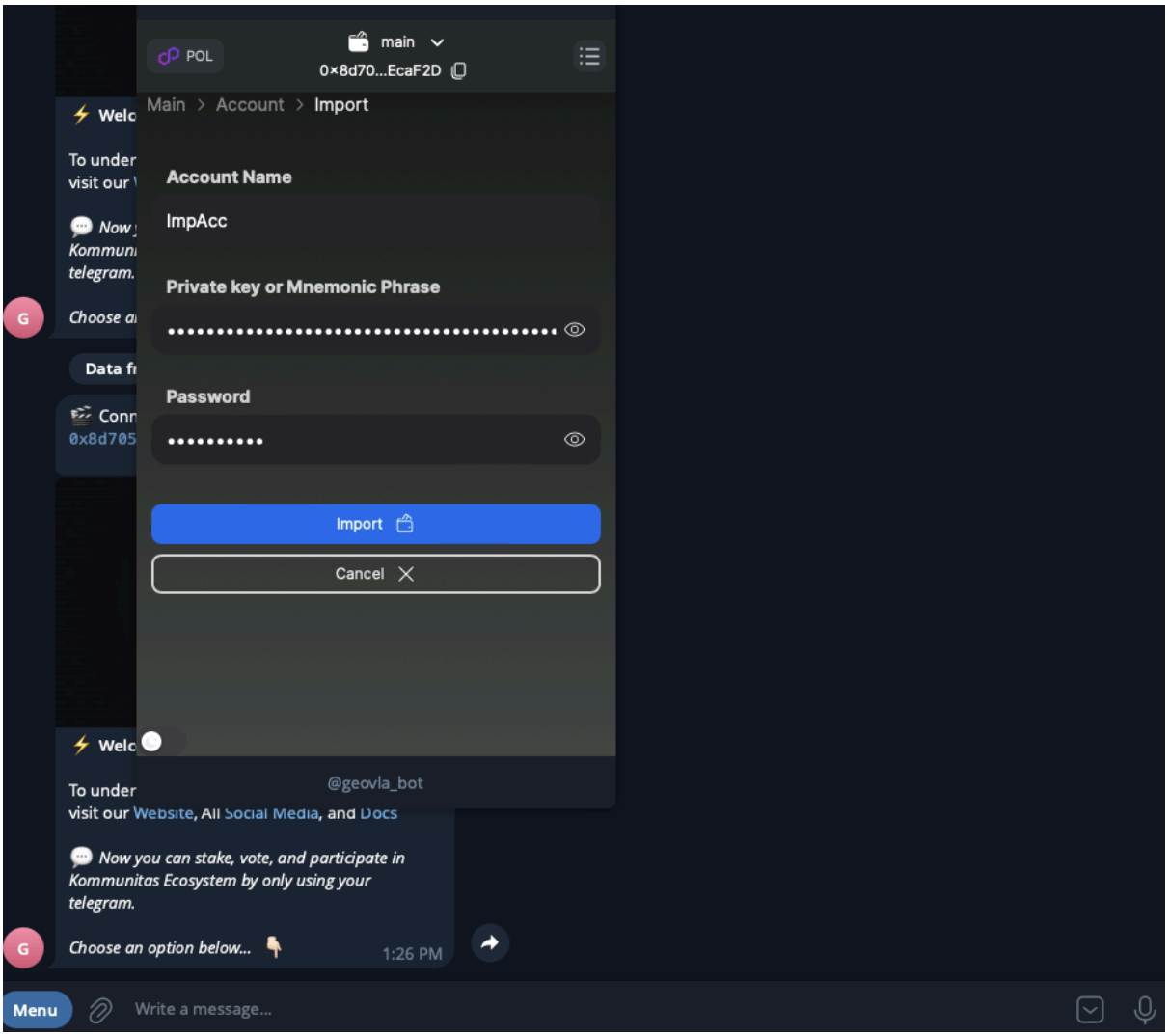
Create a new wallet

The bot guided users smoothly through the process of creating a new wallet, generating and securely storing the necessary credentials.



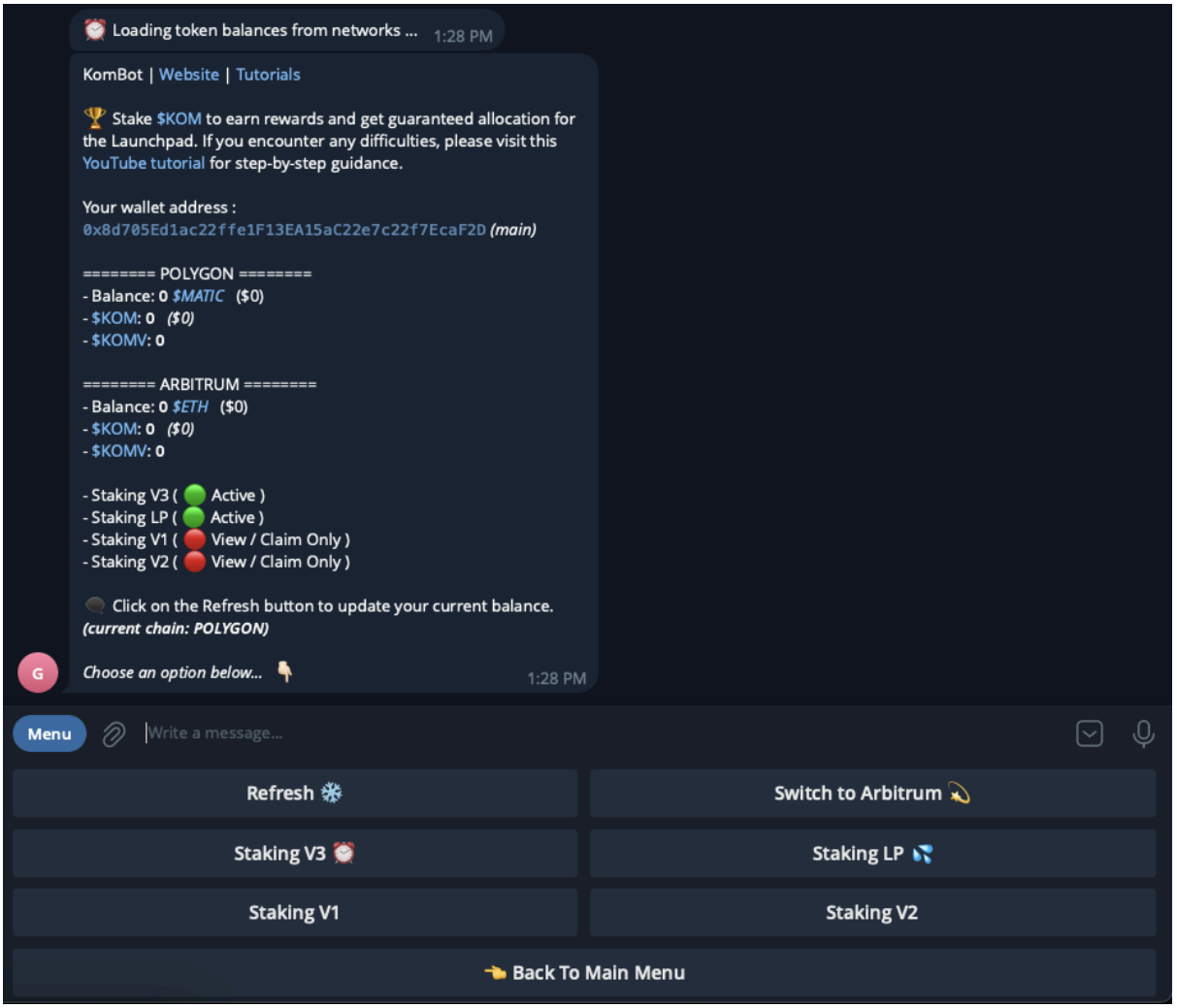
Import an existing wallet

Users were able to import existing wallets without any issues, confirming the bot's compatibility with pre-existing wallet data.



Navigate through the menu scenes

The bot's menu navigation was intuitive and responsive, allowing users to easily access different functionalities.

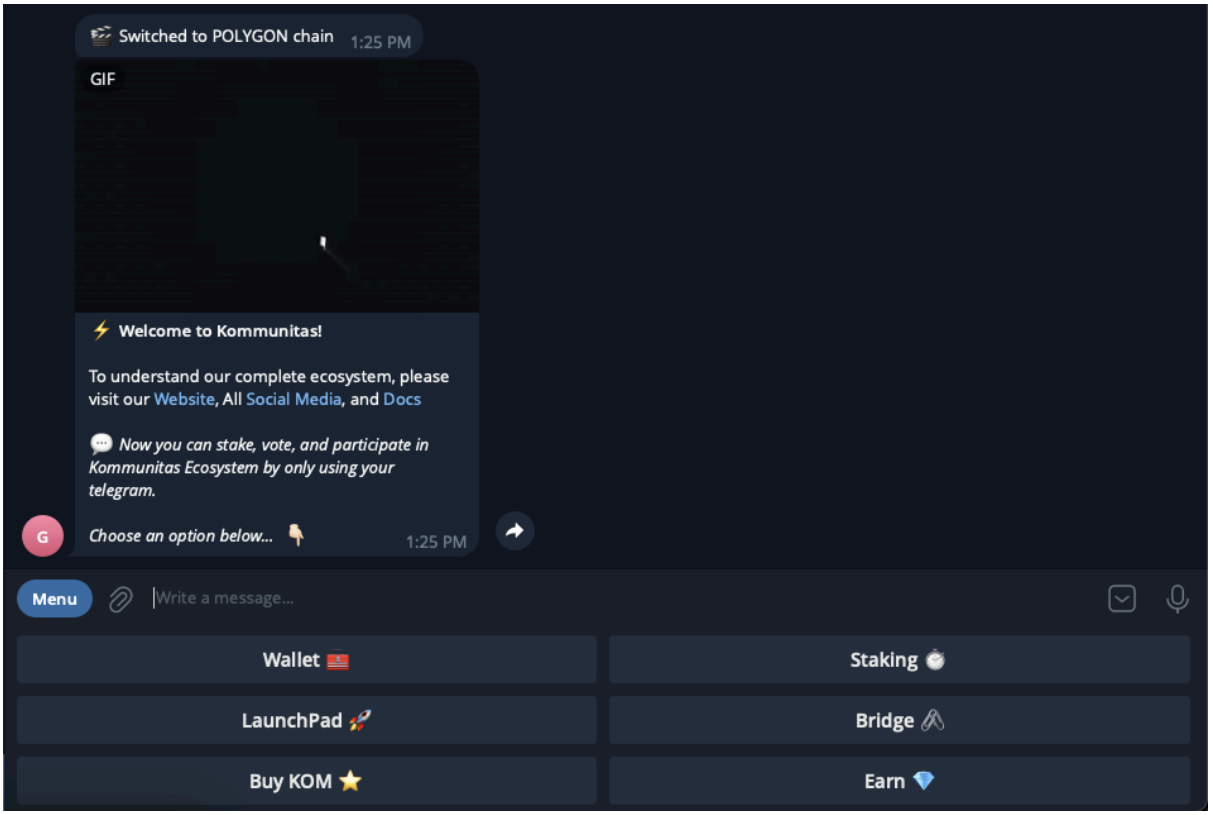


Fetch blockchain data from the Staking menu

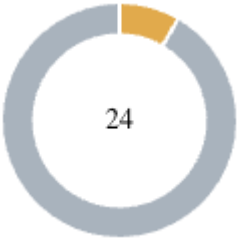
The bot successfully fetched and displayed relevant blockchain data from the Staking menu, providing users with accurate and up-to-date information.

Switched between Polygon and Arbitrum chains

The bot efficiently handled the switching between Polygon and Arbitrum chains, demonstrating its ability to manage multi-chain operations.



Findings Breakdown



Critical	0
Medium	2
Minor / Informative	22

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	22	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	EBL	Error-Prone Business Logic	Unresolved
Medium	SSB	Switch Statement Break	Unresolved
Minor / Informative	AKL	API Keys Leakage	Unresolved
Minor / Informative	BUR	Base URL Reusability	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	CLU	Console Log Usage	Unresolved
Minor / Informative	DLU	Deprecated Linter Usage	Unresolved
Minor / Informative	DRF	Duplicate Resource Fetching	Unresolved
Minor / Informative	EUOAT	Excessive Use of ANY Type	Unresolved
Minor / Informative	HAU	Hardcoded Addresses Usage	Unresolved
Minor / Informative	IRT	Inconsistent Return Types	Unresolved
Minor / Informative	IBTU	Incorrect BigInteger Type Usage	Unresolved
Minor / Informative	MFN	Misleading Function Naming	Unresolved
Minor / Informative	MEVD	Missing Environment Variables Documentation	Unresolved

●	MPC	Missing Prettier Configuration	Unresolved
●	PII	Package Import Inconsistency	Unresolved
●	RCMS	Redundant Compound Mode Selection	Unresolved
●	RCFL	Redundant Custom Formatting Logic	Unresolved
●	RTC	Redundant Type Casting	Unresolved
●	SVDC	State Variables could be Declared Constant	Unresolved
●	TED	Typescript Error Directive	Unresolved
●	UVO	Unreassigned Variables Optimization	Unresolved
●	UNP	Unused NPM Packages	Unresolved
●	USV	Unused State Variable	Unresolved

EBL - Error-Prone Business Logic

Criticality	Medium
Location	bot/controllers/main.controller.ts#L207
Status	Unresolved

Description

As part of the `messageHandler` function, the app includes a code segment that relies on the presence of specific emoji characters in `button_text` to determine which menu to return. This approach is error-prone because it depends on the correct emoji being included in the button text and can be broken easily if the emoji or button text changes.

The current approach of handling conditional logic based on text that includes emojis is used throughout the codebase. This method compromises the reliability and resilience of the application.

```
if (button_text.includes('💫')) {
  return menu_staking(ctx);
} else if (button_text.includes('🎨')) {
  return menu_staking_v3(ctx);
} else if (button_text.includes('💦')) {
  return menu_staking_lp(ctx);
}

if (ctx.message.text === '👈 BACK') {
    await ctx.scene.leave ();
    return menu (ctx);
}
```

Recommendation

To address this issue, the team is advised to use more reliable identifiers instead of relying on emojis. If possible, the button text or any other part of the app that uses such an approach should include a specific identifier or tag that can be checked. The team could consider a more robust way to identify the actions, such as additional metadata associated with the buttons.

SSB - Switch Statement Break

Criticality	Medium
Location	bot/controllers/main.controller.ts#L231
Status	Unresolved

Description

The codebase contains a switch statement with case blocks that lack `break` or `return` statements, potentially leading to fall-through behavior. These case blocks include if-else statements without an else block, which may cause issues if none of the conditions are met. Furthermore, the switch statement does not include a default case as a fallback, which is a best practice to handle unexpected values.

```
case "CANCEL_TRANSACTION":
    await ctx.reply(`😕 ${payload.message}`, { parse_mode: "HTML" });
    await ctx.scene.leave();
    if (payload.type === 'stakingv1') {
        return menu_staking_v1(ctx);
    } else if (payload.type === 'stakingv2') {
        return menu_staking_v2(ctx);
    } else if (payload.type === 'stakingv3') {
        return menu_staking_v3(ctx);
    } else if (payload.type === 'stakingLP') {
        return menu_staking_lp(ctx);
    }
case "SUCCESS_TRANSACTION":
    await ctx.reply(`🎉 ${payload.message}`, { parse_mode: "HTML" });
    await ctx.scene.leave();
    if (payload.type === 'stakingv1') {
        return menu_staking_v1(ctx);
    } else if (payload.type === 'stakingv2') {
        return menu_staking_v2(ctx);
    } else if (payload.type === 'stakingv3') {
        return menu_staking_v3(ctx);
    } else if (payload.type === 'stakingLP') {
        return menu_staking_lp(ctx);
    }
```

Recommendation

To prevent fall-through behavior and ensure the switch statement handles all cases appropriately, the team could add `break` or `return` statements to each case block and include a default case. Adding the break statements ensures each case block is exited properly, preventing unintended fall-through. Additionally, including a `default` case handles unexpected transaction types gracefully, improving the robustness of the code.

AKL - API Keys Leakage

Criticality	Minor / Informative
Location	bot/utils/utils.ts#L28,49
Status	Unresolved

Description

Hardcoding API keys in the source code is a security risk and violates best practices. These keys should be stored in environment variables to keep them secure and allow for easy configuration changes without modifying the code.

```
const headers = {
    'X-Kom-Token': '0xC004e2318722EA2b15499D6375905d75Ee5390B8',
    'accept': 'application/json'
}
const headers = {
    'x-api-key': '2eDS3r6N5KZTEdOGuKRfqVzTyuQ',
    'accept': 'application/json'
}
```

Recommendation

The team is strongly advised to move the API keys to environment variables and access them using `process.env`. By storing API keys in environment variables, the team can enhance the security of the application and make it easier to manage configurations across different environments. Additionally, the team should ensure that the `.env` file is added to `.gitignore` to prevent it from being committed to version control.

BUR - Base URL Reusability

Criticality	Minor / Informative
Location	bot/utils/utils.ts#L25,65,73,85,92,146
Status	Unresolved

Description

The `komAPI` function and the associated functions that call it (getChartData, getStatistics, getLeaderBoard) redundantly include the base URL for the Kommunitas API in multiple places. This practice can lead to errors if the base URL changes and increases the difficulty of maintaining the code.

```
export const getChartData = async (chainId: number) => {
    return komAPI(`https://api.kommunitas.net/v1/staking/chart/?chainId=${chainId}`);
}
export const getStatistics = async () => {
    return komAPI(`https://api.kommunitas.net/v1/staking/statistic`);
}
export const getLeaderBoard = async (chainId: number) => {
    return
komAPI(`https://api.kommunitas.net/v1/staking/leaderboard/?chainId=${chainId}`);
}
...
```

Recommendation

The team is recommended to set the base URL in the `komAPI` function and construct the full URL by appending the endpoint. By setting the base URL in the `komAPI` function, the code becomes easier to maintain and less error-prone, especially if the base URL changes in the future. This approach ensures that the base URL is defined in a single location, promoting DRY (Don't Repeat Yourself) principles.

CR - Code Repetition

Criticality	Minor / Informative
Location	bot/controllers/staking/lp/staking.controller.ts#L63,75,87,99 bot/controllers/staking/v3/changeCompoundMode.controller.ts#L6,48
Status	Unresolved

Description

The codebase contains repetitive code segments. There are potential issues that can arise when using repetitive code segments in JavaScript. Some of them can lead to issues like complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

For instance, the options object passed to the `reply` in the `textHandler` function is identical in all cases.

```
{
    parse_mode: 'HTML',
    reply_markup: {
        force_reply: true,
        keyboard: [
            [{ text: '👈 BACK' }],
        ],
        one_time_keyboard: true,
        resize_keyboard: true,
    }
}

const modes: Record<string, number> = { 'No Compound': 0, 'Compound My Staked $KOM
only': 1, 'Compound The Amount + Reward': 2 };
inline_keyboard: [
    [{ text: 'No Compound', callback_data: 'No Compound' }],
    [{ text: 'Compound My Staked $KOM only', callback_data: 'Compound My Staked $KOM
only' }],
    [{ text: 'Compound The Amount + Reward', callback_data: 'Compound The Amount +
Reward' }]
]
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the codebase easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the codebase. For instance, the codebase could reuse the common code segments in an internal method in order to avoid repeating the same code in multiple places.

CLU - Console Log Usage

Criticality	Minor / Informative
Location	bot/controllers/staking/index.ts#L14 bot/index.ts#L46 bot/controllers/staking/v3/main.controller.ts#L32 bot/controllers/staking/v3/staking.controller.ts#L33 lib/dbConnect.ts#L6,8
Status	Unresolved

Description

The codebase contains console logs used for debugging purposes. These logs should be removed before production deployment. If logging is necessary, it should be done through a more organized and configurable logging solution, which can handle different log levels and outputs.

```
console.log("start..");
console.log(ctx.session.account)
console.log("stakingv3 menu====>", {address});
console.log("staking====>", {address});
...
```

Recommendation

The team is advised to either remove all the unnecessary console logs or replace them with a proper logging solution. Using a proper logging solution provides better control over log levels and outputs, facilitating easier debugging and monitoring while keeping the codebase clean and professional.

DLU - Deprecated Linter Usage

Criticality	Minor / Informative
Status	Unresolved

Description

The codebase currently uses TSLint for code linting. TSLint is considered a legacy tool and its development is being phased out in favor of ESLint. The TypeScript team officially recommends migrating from TSLint to ESLint for TypeScript projects to ensure compatibility with future updates and to leverage the active development and extensive ecosystem of ESLint.

Recommendation

The team is advised to migrate the codebase from TSLint to ESLint. This involves configuring ESLint with TypeScript-specific plugins and rules to replicate the checks performed by TSLint. Migrating to ESLint ensures that the project aligns with current best practices and maintains compatibility with future TypeScript updates. This also benefits from the active community and plugin ecosystem surrounding ESLint.

DRF - Duplicate Resource Fetching

Criticality	Minor / Informative
Location	bot/controllers/staking/v3/main.controller.ts#L57,58,61,62 bot/utils/web3.ts#L72,251
Status	Unresolved

Description

The codebase concurrently fetches multiple resources using the `Promise.all` function to improve performance. However, some of these promise executions call the same function internally, leading to duplicate requests for the same resource. This is inefficient and can cause issues with APIs that have rate limits.

For instance, both the `getTokenBalances` and `getStakingV3StakedDetails` functions call the `getKOMTokenPrice()` function internally. Hence, the same resource is fetched more than once.

```
await Promise.all([
    getTokenBalances(42161, address),
    getStakingV3StakedDetails(42161, address),
    getStakershipDetails(42161, address),
    getNativeTokenPrice(42161),
    getTokenBalances(137, address),
    getStakingV3StakedDetails(137, address),
    getStakershipDetails(137, address),
    getNativeTokenPrice(137)
]);

const [nativeBalance, komBalance, komvBalance, komTokenPrice] = await Promise.all([
    getETHBalance(address, provider),
    getKOMBalance(chainId, address, provider),
    getKOMVBalance(chainId, address, provider),
    getKOMTokenPrice()
]);
const [
    { stakedAmount, stakerPendingReward },
    _userStakedLength,
    komTokenPrice
] = await Promise.all([
    _contractStaking.stakerDetail(address),
    _contractStaking.userStakedLength (address),
    getKOMTokenPrice ()
]);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that all resources are fetched only once. By doing so, the team can reduce duplicate calls and adhere to best practices for efficient API usage, preventing potential issues with rate-limited APIs.

EUOAT - Excessive Use of ANY Type

Criticality	Minor / Informative
Location	bot/controllers/main.controller.ts#L33,58,97,121,180,246 bot/controllers/staking/lp/staking.controller.ts#L9,49,115 bot/controllers/staking/v1/claim.controller.ts#L6,41,49 bot/controllers/staking/v2/claim.controller.ts#L6,40,48 bot/controllers/staking/v3/acceptStakership.controller.ts#L5,22,30 bot/controllers/staking/v3/changeCompoundMode.controller.ts#L9,59,83 bot/controllers/staking/v3/main.controller.ts#L25,141,213,261,291,307 bot/controllers/staking/v3/staking.controller.ts#L24,50,140,145 bot/controllers/staking/v3/transferStakership.controller.ts#L8,49,120 bot/controllers/staking/v3/withdraw.controller.ts#L7,48,100 bot/utils/web3.ts#L11,25,46,139 types/chains.ts#L17,21,25,29,33,37,41
Status	Unresolved

Description

The TypeScript codebase includes several instances where the `any` type is used to declare variables or parameters. While TypeScript provides the flexibility to use the `any` type when the type is not precisely known or needs to be intentionally left open, excessive use of `any` can undermine the benefits of static typing. Overusing `any` diminishes the advantages of TypeScript, as it removes the benefits of type checking and type safety, making the code more error-prone and less maintainable. The following code segments are a sample from all the occurrences.

```
ctx: any
provider: any
abi: any
...
```

Recommendation

The team is advised to replace `any`, whenever possible, with precise types to provide clarity about the expected data structure or format. By creating explicit type definitions for objects, functions, and parameters, the team could ensure robust type checking and improved code readability. Union types or generics could be used to handle scenarios where a variable can have multiple types, maintaining the advantages of type safety. By minimizing the use of the `any` type and leveraging TypeScript's strong typing features, the codebase can benefit from improved developer experience, better IDE support, and enhanced code maintainability.

HAU - Hardcoded Addresses Usage

Criticality	Minor / Informative
Location	bot/controllers/staking/lp/main.controller.ts#L23,24,25,31 bot/controllers/staking/v1/main.controller.ts#L24 bot/controllers/staking/v2/main.controller.ts#L25 bot/utils/utils.ts#L28 bot/controllers/staking/lp/staking.controller.ts#L27
Status	Unresolved

Description

The application includes a `CONTRACTS` config object that summarizes all the addresses and ABIs for the contracts. However, some of these addresses are still hardcoded directly in the codebase instead of using the centralized config object. This inconsistency can lead to errors, makes the code harder to maintain, and increases the risk of using outdated addresses.

```
const headers = {
    'X-Kom-Token': '0xC004e2318722EA2b15499D6375905d75Ee5390B8',
    'accept': 'application/json'
};
const msg =
    `KomBot | <a href="https://staking.kommunitas.net/">Website</a> | <a
href='https://youtu.be/CkdGN54ThQI?si=1RZ0T531IeMGfgaQ'>Tutorials</a>\n\n` +
    `<b>💎 Staked :</b>  <b>${_balance}</b> <i><a
href='https://polygonscan.com/address/0xC004e2318722EA2b15499D6375905d75Ee5390B8'>$KO
M</a></i>` +
    `\n\n⚠ <i>StakingV2 Pool has been closed.</i>`;
...
```

Recommendation

To address this issue, the team could refactor the code to ensure all contract addresses and ABIs are referenced from the `CONTRACTS` config object. This ensures that all addresses and ABIs are centralized and easier to update and manage. By centralizing all contract addresses and ABIs, the codebase becomes more maintainable and secure. This approach reduces the risk of using incorrect or outdated addresses and makes it easier to update addresses when necessary.

IRT - Inconsistent Return Types

Criticality	Minor / Informative
Location	bot/utils/web3.ts#L46,62,236,275,295
Status	Unresolved

Description

The codebase includes certain functions with inconsistent return types. For instance, the `getKOMVBalance` function returns a string when successful and a number when it fails. This inconsistency can lead to bugs and difficulties in handling the function's output, as the caller must account for multiple return types.

```
export const getKOMVBalance = async (chainId: number, address: string, provider: any)
=> {
    try {
        const { address: CONTRACT_ADDRESS, abi } = CONTRACTS[chainId].KOMV;
        const contract = new ethers.Contract(CONTRACT_ADDRESS, abi, provider);
        const _balance = await contract.balanceOf(address);
        return ethers.utils.formatUnits (_balance, 0);
    } catch (err) {
        return 0.0;
    }
}
```

Recommendation

To address this issue, the team is strongly advised to ensure all functions return a consistent type. Alternatively, if the team needs to distinguish between successful and failed calls, the team can return an object or array that includes both the result and an error (if it fails). Using consistent return types or a structured object helps improve code reliability and simplifies handling of the function's output.

IBTU - Incorrect BigInteger Type Usage

Criticality	Minor / Informative
Location	bot/utils/web3.ts#L13
Status	Unresolved

Description

The codebase incorrectly uses `BigInt` as a type instead of the correct lowercase `bigint`. In TypeScript, bigint is the correct type annotation for BigInt values, while BigInt is the constructor for creating BigInt values. Using the incorrect type can lead to type errors and confusion.

```
const _balance: BigInt = await provider.getBalance(address);
```

Recommendation

To address this issue the team is recommended to replace `BigInt` with `bigint` to correctly type annotate the variable. Using `bigint` ensures that the variable is correctly typed, improving type safety and code clarity. This change aligns with TypeScript best practices and helps prevent potential type-related issues.

MFN - Misleading Function Naming

Criticality	Minor / Informative
Location	bot/utils/staking.ts#L10
Status	Unresolved

Description

Functions can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The application uses some function names that are too generic or do not clearly convey the underneath functionality. Misleading function names can lead to confusion, making the code more difficult to read and understand.

```
export const calculateReard = async (stakingAmount: number, stakingPeriod: number) =>
{
    try {
        const _chain = chains[137];
        const { address: STAKING_CONTRACT_ADDRESS, abi: STAKING_ABI } =
CONTRACTS[137].STAKING_V3;
        const provider = new ethers.providers.JsonRpcProvider(_chain.rpc);
        const _contractKOM = new ethers.Contract(STAKING_CONTRACT_ADDRESS,
STAKING_ABI, provider);
        const _reward = await _contractKOM.calculateReward(stakingAmount * 1e8,
stakingPeriod);
        return ethers.utils.formatUnits(_reward, 8);
    } catch (err) {
        return '0.0';
    }
}
```

Recommendation

It's always a good practice for the codebase to contain function names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEVD - Missing Environment Variables Documentation

Criticality	Minor / Informative
Status	Unresolved

Description

The codebase is lacking proper documentation for the required environment variables. Environment variables are crucial for configuring various aspects of the application, such as API keys, database connection strings, or other sensitive information. Without clear documentation, it becomes challenging for developers and other users to understand which environment variables are required, their purpose, the expected format, and how to set them up.

```
const _bot = new Telegraf(process.env.BOT_TOKEN, {
  handlerTimeout: 9_000_000, // 2.5 hours in milliseconds
});
Markup.button.webApp("Connect Wallet 🧰",
`${process.env.MINIAPP_URL}?chainId=${chainId}&forWalletConnection=true`)
mongoose.connect(process.env.mongoURI as string);
...
```

Recommendation

The team is advised to provide comprehensive documentation for all required environment variables in the codebase. By providing thorough documentation for environment variables, you improve the maintainability, collaboration, and security of the app. Developers and operators will have a clear understanding of the required configurations, which leads to fewer errors and smoother deployments. Additionally, the team could add a `.env.example` file in the codebase to list all the required variables.

MPC - Missing Prettier Configuration

Criticality	Minor / Informative
Status	Unresolved

Description

The app lacks a Prettier configuration file, which results in inconsistent code formatting. The codebase exhibits inconsistencies in spacing and comments formatting in several places. Without a Prettier configuration, contributors may use different formatting styles, leading to a codebase that is harder to read and maintain.

```
// staking comds
stakingCommands (_bot);
// main comds
mainCommands (_bot);
```

Recommendation

The team is advised to add a Prettier configuration file (`.prettierrc` or `prettier.config.js`) to the root of the project. This ensures that all contributors follow the same formatting guidelines. Additionally, the team could consider adding a prettier script to the `package.json` to facilitate formatting. By enforcing a consistent coding style, the codebase will become more readable and maintainable, improving overall code quality.

PII - Package Import Inconsistency

Criticality	Minor / Informative
Location	bot/controllers/staking/v3/main.controller.ts#L20 bot/controllers/staking/v3/transferStakership.controller.ts#L7 bot/utils/staking.ts#L3 bot/utils/web3.ts#L4
Status	Unresolved

Description

The codebase is configured to use ECMAScript Module (ESM) syntax for imports. However, certain files still use the legacy `require` statement to import packages, leading to inconsistent import practices. This can cause confusion and potential issues with module loading.

```
import { chains, CONTRACTS } from "../../constants/config";
import { STAKEV3_DETAIL_ITEM } from "@/types";
import { getKOMTokenPrice } from "./utils";
const { ethers } = require('ethers');

import { chains } from "../../constants/config";
import { CONTRACTS } from "../../constants/config";
const { ethers } = require('ethers');
...
```

Recommendation

The team is advised to refactor the code to consistently use the import syntax for all module imports and ensure that all imports use the import syntax to maintain consistency throughout the codebase. This aligns with modern JavaScript standards and improves code readability and maintainability.

RCMS - Redundant Compound Mode Selection

Criticality	Minor / Informative
Location	bot/controllers/staking/v3/changeCompoundMode.controller.ts#L91
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

As part of the `callbackQuery` function, the app checks if the user selects the same compound mode as before and prompts them to select a different mode. This can be simplified by disabling the button for the current mode, which would prevent the user from selecting it in the first place. This approach improves the user experience and reduces unnecessary conditions in the code.

```
if (compoundType === newCompoundType) {
    ctx.reply(
        `⚠ You have selected the same compound mode as before.\nPlease select a
different mode.`,
        {
            reply_markup: {
                keyboard: [
                    [{ text: '👈 BACK' }],
                ],
                one_time_keyboard: true,
                resize_keyboard: true,
            }
        }
    );
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RCFL - Redundant Custom Formatting Logic

Criticality	Minor / Informative
Location	bot/utils/utils.ts#L220
Status	Unresolved

Description

The function `formatNumber` manually formats a number with commas and restricts decimal places. This custom logic is complex, error-prone, and less maintainable compared to using built-in JavaScript features. The Intl.NumberFormat API can simplify this task.

```
export const formatNumber = (number: number | string | unknown | bigint, len = 4) =>
{
    if (Number(number) === 0 && isNaN(Number(number))) return 0;
    let [num, _decimal] = String(number).split(".");
    let _num = "";
    let j = 1;
    for (let i = num.length - 1; i >= 1; i--, j++) {
        _num += num[i];
        if (j % 3 === 0) _num += ",";
    }
    _num += num[0];
    let str = _num.split("").reverse().reduce((acc: string, item: string) => acc +=
item, "");
    if (_decimal) str += `.${_decimal.substring(0, 2)}`;

    return str;
};
```

Recommendation

The team is strongly recommended to use the Intl.NumberFormat API to format numbers. Using Intl.NumberFormat provides a more reliable and maintainable way to format numbers, leveraging built-in internationalization features and reducing the complexity of the code.

RTC - Redundant Type Casting

Criticality	Minor / Informative
Location	bot/utils/utils.ts#L182,183
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The code casts the number variable to `Number` twice, which is redundant and less efficient. It first checks if number is NaN after converting it to a number, and then converts it again to pass it to `Math.floor` .

```
if (isNaN(Number(number))) throw "0";
const num = Math.floor(Number(number));
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

SVDC - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	bot/utils/web3.ts#L225,226 bot/index.ts#L28 bot/controllers/main.controller.ts#L35,100
Status	Unresolved

Description

State variables can be declared as constant using the `const` keyword and initialized at the top of the file or a separate one. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables improve performance and memory consumption.

Furthermore, there are certain static variables declared as `const` inside function scopes. Hence, these variables will be created every time the functions are called.

The following segments is a sample of all the occurences.

```
return {
    originStakership: '0x0000000000000000000000000000000000000000',
    pendingStakership: '0x0000000000000000000000000000000000000000',
}

const commands = [
  { command: '/start', description: 'Start a dialogue with the bot' },
  { command: '/menu', description: 'Show Main Menu' },
  { command: '/menu_staking', description: 'Show Menu for Stakings' },
  { command: '/menu_staking_v1', description: 'Show Menu for Staking V1' },
  { command: '/menu_staking_v2', description: 'Show Menu for Staking V2' },
  { command: '/menu_staking_v3', description: 'Show Menu for Staking V3' },
  { command: '/menu_staking_lp', description: 'Show Menu for Staking LP' },
  { command: '/launchpad', description: 'Show lanunchpad menu' },
  { command: '/help', description: 'Get help and instructions' },
];
const msg = `⚠ No connected wallet!\n\nPlease connect wallet first... 👇`;
const message =
    `<b>⚡ Welcome to Kommunitas!</b>\n\n` +
    `To understand our complete ecosystem, please visit our <a
href='https://www.kommunitas.net/'>Website</a>, All <a
href='https://linktr.ee/kommunitas'>Social Media</a>, and <a
href='https://docs.kommunitas.net'>Docs</a>` +
    `\n\n<i>💬 Now you can stake, vote, and participate in Kommunitas Ecosystem by
only using your telegram.</i>\n` +
    `\n<i>Choose an option below...</i>  👇`;
```

Recommendation

Constant state variables can be useful when the codebase wants to ensure that the value of a state variable cannot be changed by any function. This can be useful for storing values that are important to the app's behavior. The team is advised to add the `const` keyword to state variables that never change. Additionally, the team could move static constant variables which are declared inside function scopes to the most upper scope possible, so that they are declared only once.

TED - Typescript Error Directive

Criticality	Minor / Informative
Location	bot/index.ts#L21
Status	Unresolved

Description

The codebase uses the `@ts-ignore` directive to suppress TypeScript errors. However, `@ts-ignore` will silently do nothing if the following line of code is error-free, potentially masking unintended issues. The `@ts-expect-error` directive is more appropriate as it will alert you if the line of code does not produce an error, ensuring that you only suppress expected errors.

```
//@ts-ignore
const stages = new Scenes.Stage([stakingV3Scene, withdrawV3Scene,
transferStakershipScene, acceptStakershipScene, changeCompoundModeScene,
stakingLPScene, claimWithV1Scene, claimWithV2Scene]);
```

Recommendation

The team is advised to replace `@ts-ignore` with `@ts-expect-error` to ensure that you are only suppressing expected errors. Using `@ts-expect-error` makes the intention clear that an error is expected and should be ignored. If the line of code is error-free, TypeScript will generate a warning, helping to maintain code quality and catching potential issues early.

UVO - Unreassigned Variables Optimization

Criticality	Minor / Informative
Location	bot/controllers/staking/lp/main.controller.ts#L21 bot/controllers/staking/v3/main.controller.ts#L270 bot/utils/utils.ts#L222
Status	Unresolved

Description

The code initializes certain variables using `let` but never reassigns their value throughout their scope. Since the variables are not intended to be reassigned, it's recommended to use `const` for better code clarity and to convey the intent that the value remains constant.

```
let cnt = 0;
let [num, _decimal] = String(number).split(".");
```

Recommendation

The team is advised to modify the declaration of these variables from `let` to `const` since their value is not reassigned. This change not only reflects the intended immutability of the variables but also enhances code readability by signaling to developers that the value should not be changed after initialization. Using `const` for variables that don't need reassignment is a good practice in maintaining clean and understandable code.

UNP - Unused NPM Packages

Criticality	Minor / Informative
Location	package.json#L29,30,31,32
Status	Unresolved

Description

The codebase includes several npm packages that are not being used. This can lead to bloated `node_modules`, longer build times, and potential security vulnerabilities.

```
"express": "^4.19.2",
"express-validator": "^7.0.1",
"jsonwebtoken": "^9.0.2",
"mongoose": "^8.3.0",
```

Recommendation

The team is recommended to review the codebase to confirm that these packages are indeed unused. If they are not required, the team could uninstall these packages from the codebase. By removing unused packages, the team can reduce the attack surface for potential vulnerabilities, decrease the size of `node_modules`, and improve overall project maintainability. This practice ensures that the project dependencies are lean and only include necessary modules.

USV - Unused State Variable

Criticality	Minor / Informative
Location	bot/commands/main.commands.ts#L8,249,253 bot/controllers/staking/lp/staking.controller.ts#L22,115 bot/controllers/staking/lp/main.controller.ts#L18 bot/controllers/staking/v1/claim.controller.ts#L18,49 bot/controllers/staking/v2/claim.controller.ts#L18,48 bot/controllers/staking/v3/main.controller.ts#L45,239,270,291 bot/scenes/staking/v3/acceptStakership.scene.ts#L1 bot/controllers/staking/v3/changeCompoundMode.controller.ts#L64 bot/controllers/staking/v3/transferStakership.controller.ts#L77 bot/controllers/staking/v3/withdraw.controller.ts#L27,54 bot/utils/utils.ts#L7,8,220
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the codebase, but is never used in any of the codebase's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the codebase and make it more difficult to understand and maintain. They can also increase the size of the codebase and impact the performance of the app.

```
Markup
idx
cnt
err
...
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the app's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the codebase.

Summary

This report provides a thorough assessment of the application's security and performance. Through meticulous analysis, the report identifies vulnerabilities and weaknesses in key areas such as data handling and network security. Recommendations are provided to address these issues and enhance the application's resilience against cyber threats.

Overall, the report serves as a valuable resource, offering insights into the application's security posture and actionable recommendations to fortify its defenses. By implementing the suggested measures, the team can strengthen the app’s security foundation and maintain trust among users.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io