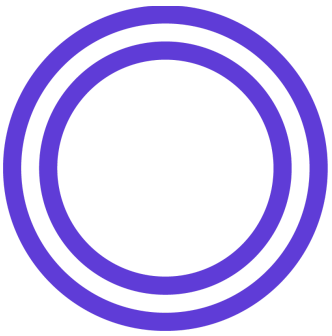
Saros DLMM

Smart Contract Audit Report
Prepared for Coin98



Date Issued:	May 20, 2025
Project ID:	PI250064
Version:	v1.0
Confidentiality Level:	Public

RECONIX

Report Information

Project ID	PI250064
Version	v1.0
Client	Coin98
Project	Saros DLMM
Auditor(s)	Ronnachai Chaipha Peeraphut Punsuwan Wachirawit Kanpanluk
Author(s)	Ronnachai Chaipha Peeraphut Punsuwan Wachirawit Kanpanluk
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	May 20, 2025	Full report	Ronnachai Chaipha Peeraphut Punsuwan Wachirawit Kanpanluk

Contact Information

Company	Reconix
Phone	(+66) 90 888 7186
Email	contact@reconix.co

Table of Contents

1. Executive Summary

As requested by Coin98, Reconix team conducted an audit to verify the security posture of the Saros DLMM smart contracts between April 28, 2025 and May 9, 2025. During the audit, Reconix team examined all smart contracts and the overall operation within the scope to understand the overview of Saros DLMM smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Reconix found 1 medium, and 2 very low-severity issues. With the project team's prompt response, 1 medium issue was mitigated in the reassessment, while 2 very low-severity issues were acknowledged by the team. However, as the source code is currently not publicly available, there is a potential risk that the smart contracts deployed on the blockchain may not be identical to the audited smart contracts. This discrepancy could result in introducing security vulnerabilities or unintended behaviors that were not identified during the audit process. It is of importance to recognize that interacting with an unverified smart contract may lead to the potential loss of funds.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Reconix suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Saros DLMM is a next-generation AMM primitive based on the Liquidity Book. It is designed to make liquidity capital-efficient, composable, and resilient by default across a wide range of assets, market conditions, and use cases.

Scope Information:

Project Name	Saros DLMM
Website	https://www.saros.xyz
Smart Contract Type	Solana Smart Contract
Chain	Solana
Programming Language	Rust
Category	DEX

Audit Information:

Audit Method	Whitebox
Audit Date	April 28, 2025 - May 9, 2025
Reassessment Date	May 20, 2025

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited by Reconix in detail:

Initial Audit: (Commit: 3ff57480a7baac6b53247645e6dadca2e66439b1)

Contract	Location (URL)
liquidity-book	https://github.com/saros-xyz/saros-sol-liquidity-book/blob/3ff57480a7/programs/liquidity-book

Reassessment Audit: (Commit: -)

No reassessment commit was provided because the client provided the clarification and acknowledged the issues.

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

2.3. Security Model

2.3.1 Trust Modules

The Saros DLMM has certain privileged roles with the authority to mutate the critical state variables of the contract. Changes to these state variables significantly impact the contract's functionality. The privileged roles and their corresponding privileged functions are enumerated as follows:

- The `preset_authority` address can perform the following actions:
  - Transfer `preset_authority` privileged roles to another account.
  - Setting the platform configuration.
- The `deployer` address can upgrade the contract implementation; this privileged operation could be used to drain all funds in the contract.

The Saros DLMM several functionalities have relied on the external components, which may significantly impact the contract if they malfunction. The external components are listed as follows:

- The `hook-interface` program is an external program that contains the logic to be executed before and after transferring tokens. This exposes the risk to all funds.
- The `mdma-hook` program is responsible for distributing rewards for each position and managing the reward distribution settings for users. This can impact the reward belonging to all users.

2.3.2 Trust Assumptions

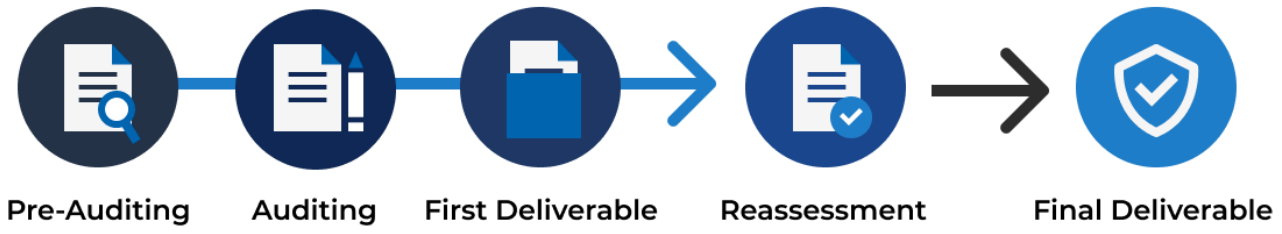
In the Saros DLMM, the protocol's privileged roles, have the ability to change the critical state variables of the contract, also external components such were assumed to be trusted. Acknowledging these trust assumptions is important, as it introduces substantial risks to the platform. Trust assumptions include, but are not limited to:

- All privileged roles perform the privileged function with good will.
- The `deployer` address is trusted to upgrade the contract implementation.
- The `preset_authority` address is trusted to grant only the proper address as new `preset_authority`.
- The `hook-interface` program is assumed to work correctly all the time.
- The `mdma-hook` program is assumed to work correctly all the time.

3. Methodology

Reconix conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Reconix smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/ReconixCo/SCSTG/releases/download/v1.0/SCSTG.pdf) which covers most prevalent risks in smart contracts.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Testing Arithmetic Operation and Conversion	1.1. Integer Overflow and Underflow 1.1.1. Solidity compiler version 0.8.0 and higher 1.1.2. Solidity compiler version 0.8.0 and below 1.2. Precision Loss 1.2.1. The rounding down of the division 1.2.2. The order of division and multiplication 1.3. Type conversion 1.3.1. The change of size (Same type with different size conversion) 1.3.2. The change of type (Different types with the same size conversion) 1.3.3. The change of sign (Different sign conversion)
2. Testing Contract Compiling	2.1. Contract dependency 2.1.1. Contract implementation should comply with the standards specification 2.1.2. Built-in symbols should not be shadowed 2.2. Solidity 2.2.1. Solidity compiler version should be specific 2.2.2. State and function visibility should be explicitly labeled 2.2.3. Functions that are never called internally should not have public visibility
3. Testing External Interaction	3.1. Invoking external calls 3.1.1. Unknown external components should not be invoked 3.1.2. Delegatecall should not be used on untrusted contracts 3.1.3. Invoke function with “this” keyword should be used with caution
4. Testing Privilege Function	4.1. Privilege functions 4.1.1. State variables should not be unfairly controlled by privileged accounts 4.1.2. Privileged functions or modifications of critical states should be logged
5. Testing Control Flow	5.1. Reentrancy 5.1.1 Reentrant calling should not negatively affect the contract states

Testing Category	Testing Items
	5.2. Input validation 5.2.1. Lack of input validation
6. .Testing Access Control	6.1. Contract's authentication 6.1.1. tx.origin should not be used for authentication 6.1.2. Authentication measures must be able to correctly identify the user 6.2. Contract's authorization 6.2.1. The roles are well defined and enforced 6.2.2. The roles can be safely transferred 6.2.3. Least privilege principle should be used for the rights of each role 6.3. Signature verification 6.3.1. Signed signature should be used properly 6.4. Access control on critical function 6.4.1. The critical function should enforce an access control
7. Testing Randomness	7.1. External Source 7.1.1. VRF 7.1.2. Provenance hash 7.2. Internal Source 7.2.1. Future block hash
8. Testing Loop Operation	8.1. Block gas limit 8.1.1. Gas cost could exceed the block limit from loop operations 8.2. Reusing msg.value 8.2.1. Improper using msg.value in a loop 8.3. Unexpected revert inside loop 8.3.1. Using multiple external calls in a loop 8.4. Using flow control expressions over loop execution 8.4.1. Control flow operator skips a crucial part of code 8.5. Inconsistent loop iterator 8.5.1. Having multiple expressions that alter the same iterator of the loop 8.5.2. Variable loop boundary
9. Testing Contract Upgradability	9.1. Identify an upgradability in contract 9.1.1. Identify a delegatecall instruction that could lead to the contract upgradability 9.1.2. Identify a selfdestruct instruction that could lead to the contract upgradability 9.2. The initialize function implementation 9.2.1. The initialize function could only be executed once by the authorized party 9.3 Upgradable proxy contract pitfalls

Testing Category	Testing Items
	9.3.1. Storage slot allocation should not conflict

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

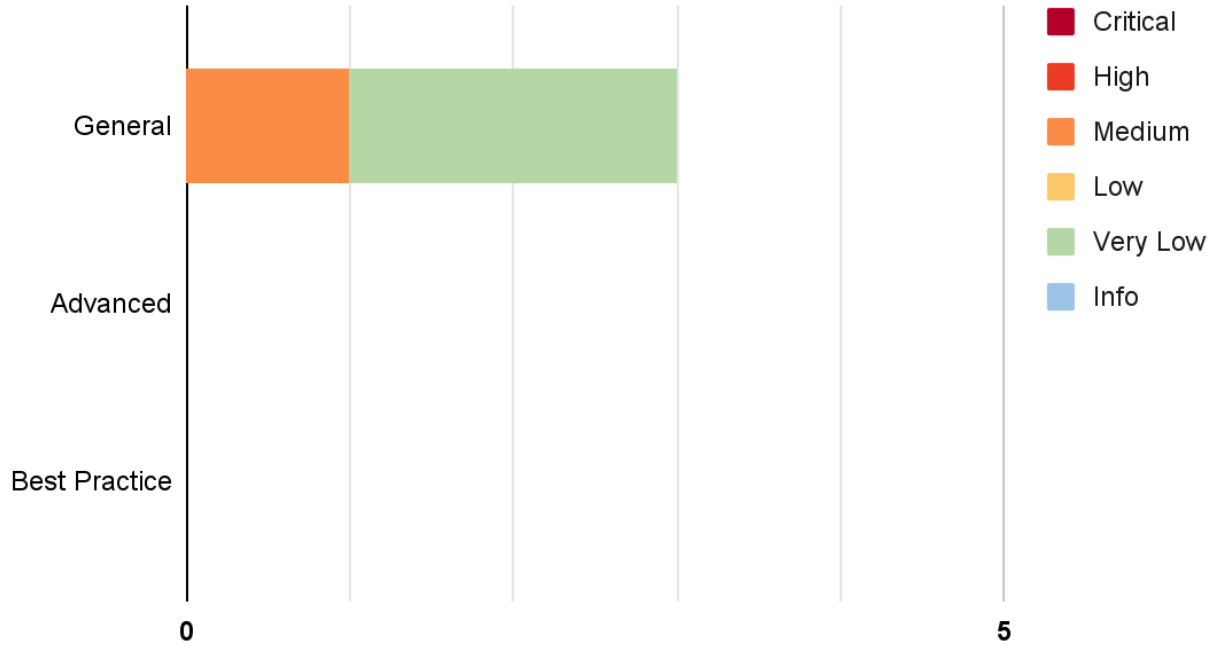
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

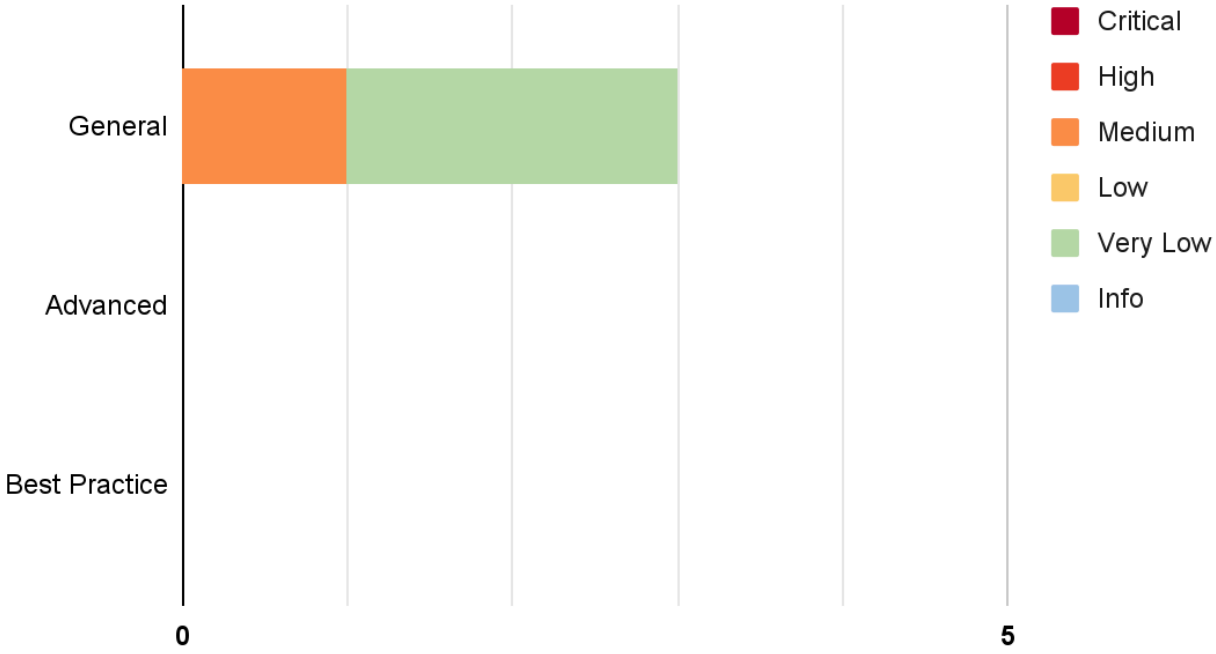
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
ID-001	Centralized Authority Control	General	Medium	Resolved *
ID-002	Use of Outdated Dependency	General	Very Low	Acknowledged
ID-003	Insufficient Logging for Privileged Functions	General	Very Low	Acknowledged

* The mitigations or clarifications by Coin98 can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Authority Control

ID	ID-001
Target	liquidity-book
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** High The controlling authorities have the ability to manipulate critical state variables, altering the contract's behavior and While the contract is intended for internal use with a limited number of participants, any unauthorized or accidental changes to critical parameters. **Likelihood:** Low The private key of the controlling authorities is unlikely to be compromised. Nevertheless, if it were to be compromised, there are no restrictions preventing unauthorized changes from being made.
Status	Resolved * The Coin98 team has mitigated this issue by confirming that they will use the multisig account as an authorized party to ensure that all privilege contracts are well prepared since the multisig account's execution requires that a list of members in the authorized party must agree.

5.1.1. Description

The contract allows authorized roles, such as the `deployer` address, to upgrade the contract implementation, and the `preset_authority` address to update critical parameters without any control mechanisms. If the private key of a controlling authority is compromised, there are no restrictions preventing unauthorized changes from being made.

File	Function	Role
programs/liquidity-book/src/lib.rs(L:22)	initialize_bin_step_config()	preset_authority
programs/liquidity-book/src/lib.rs(L:40)	initialize_quote_asset_badge()	preset_authority
programs/liquidity-book/src/lib.rs(L:87)	update_bin_step_config()	preset_authority
programs/liquidity-book/src/lib.rs(L:96)	update_quote_asset_badge()	preset_authority

File	Function	Role
programs/liquidity-book/src/lib.rs(L:103)	update_pair_static_fee_parameters()	preset_authority
programs/liquidity-book/src/lib.rs(L:110)	withdraw_protocol_fees()	preset_authority
programs/liquidity-book/src/lib.rs(L:114)	transfer_config_ownership()	preset_authority
programs/liquidity-book/src/lib.rs(L:125)	set_hook()	preset_authority

5.1.2. Recommendation

In the ideal case, the critical state variables should not be modifiable to preserve the integrity of the smart contract. However, if modification is necessary due to internal usage, Reconix suggests using a multisig mechanism to improve transparency and reduce the likelihood of the controlling authorities' private key being compromised.

5.2. Use of Outdated Dependency

ID	ID-002
Target	liquidity-book
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low Outdated dependencies have publicly known issues and bugs. It is possible that attackers can use those flaws to attack the program and cause monetary loss or business impact to the platform and its users. **Likelihood:** Low With the current dependency version, it is very unlikely that the publicly known bugs and issues will affect the target program.
Status	**Acknowledged** The Coin98 team clarified that the program is only used for testing.

5.2.1. Description

A dependency used by the `liquidity-book` is outdated. The version may have publicly known inherent bugs that may potentially be used to cause damage to the program or the users of the program.

Cargo.toml

```
[workspace]
members = ["programs/*"]
resolver = "2"

[workspace.dependencies]
# Anchor
anchor-lang = { version = "0.30.1", features = ["event-cpi"] }
anchor-spl = "0.30.1"

# Program deps
bytemuck = { version = "1.16.1", features = ["derive", "min_const_generics"] }
ruint = "1.9.0"

# Test utils
joelana-math = { path = "crates/program-utils/math" }
joelana-tokens = { path = "crates/program-utils/tokens" }
joelana-test-utils = { path = "crates/test-utils" }

```

```
19 # EVM tools
20 revm = { version = "12.1.0", features = ["optional_balance_check"] }
21 alloy = { version = "0.2.0", features = ["sol-types", "dyn-abi", "json"] }
22
```

The `alloy` crate used is outdated. The version specified in the program is `0.2.0`; however, the latest stable version at the time of the audit is `0.2.1`.

5.2.2. Recommendation

Reconix suggests upgrading the dependency to the latest stable version. However, the compatibility of the components must be checked to make sure that the program is functional as intended.

5.3. Insufficient Logging for Privileged Functions

ID	ID-003
Target	liquidity-book
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is unlikely that the execution of the privileged functions will be a malicious action.
Status	**Acknowledged** The Coin98 team clarified that the issue will be fixed along with the hook's code completion.

5.3.1. Description

Critical functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform. The `handler()` privilege function does not emit any event as shown below:

programs/liquidity-book/src/instructions/admin/set_hook.rs

```
26 pub fn handler<'c: 'info, 'info>(ctx: Context<'_, '_, 'c, 'info,
   SetHook<'info>>) -> Result<()> {
27     let pair = &mut ctx.accounts.pair;
28     let hook = &ctx.accounts.hook;
29
30     pair.hook = Some(hook.key());
31
32     hook_interface::cpi::on_hook_set(
33         CpiContext::new_with_signer(
34             ctx.accounts.hooks_program.to_account_info(),
35             HookInvocation {
36                 hook: ctx.accounts.hook.to_account_info(),
37                 pair: ctx.accounts.pair.to_account_info(),
38             },
39             &[&ctx.accounts.pair.seeds()],
40         )
41         .with_remaining_accounts(ctx.remaining_accounts.to_vec()),
42     )?;
```

```
43
44      Ok(())
45  }
```

5.3.2. Recommendation

Reconix suggests emitting events for the execution of critical functions, for example:

programs/liquidity-book/src/instructions/admin/set_hook.rs

```
26  pub fn handler<'c: 'info, 'info>(ctx: Context<'_, '_, 'c, 'info,
    SetHook<'info>>) -> Result<()> {
27      let pair = &mut ctx.accounts.pair;
28      let hook = &ctx.accounts.hook;
29
30      pair.hook = Some(hook.key());
31
32      hook_interface::cpi::on_hook_set(
33          CpiContext::new_with_signer(
34              ctx.accounts.hooks_program.to_account_info(),
35              HookInvocation {
36                  hook: ctx.accounts.hook.to_account_info(),
37                  pair: ctx.accounts.pair.to_account_info(),
38              },
39              &[&ctx.accounts.pair.seeds()],
40          )
41          .with_remaining_accounts(ctx.remaining_accounts.to_vec()),
42      )?;
43
44      emit_cpi!(LiquidityBookConfigSetHookEvent {
45          pair: pair.key(),
46          hook: hook.key()
47      });
48
49      Ok(())
50  }
```

6. Appendix

6.1. About Reconix

REALISTIC SOLUTIONS FOR YOUR CYBERSECURITY NEEDS

Reconix is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://reconix.co/
Facebook	https://www.facebook.com/ReconixCo


RECONIX