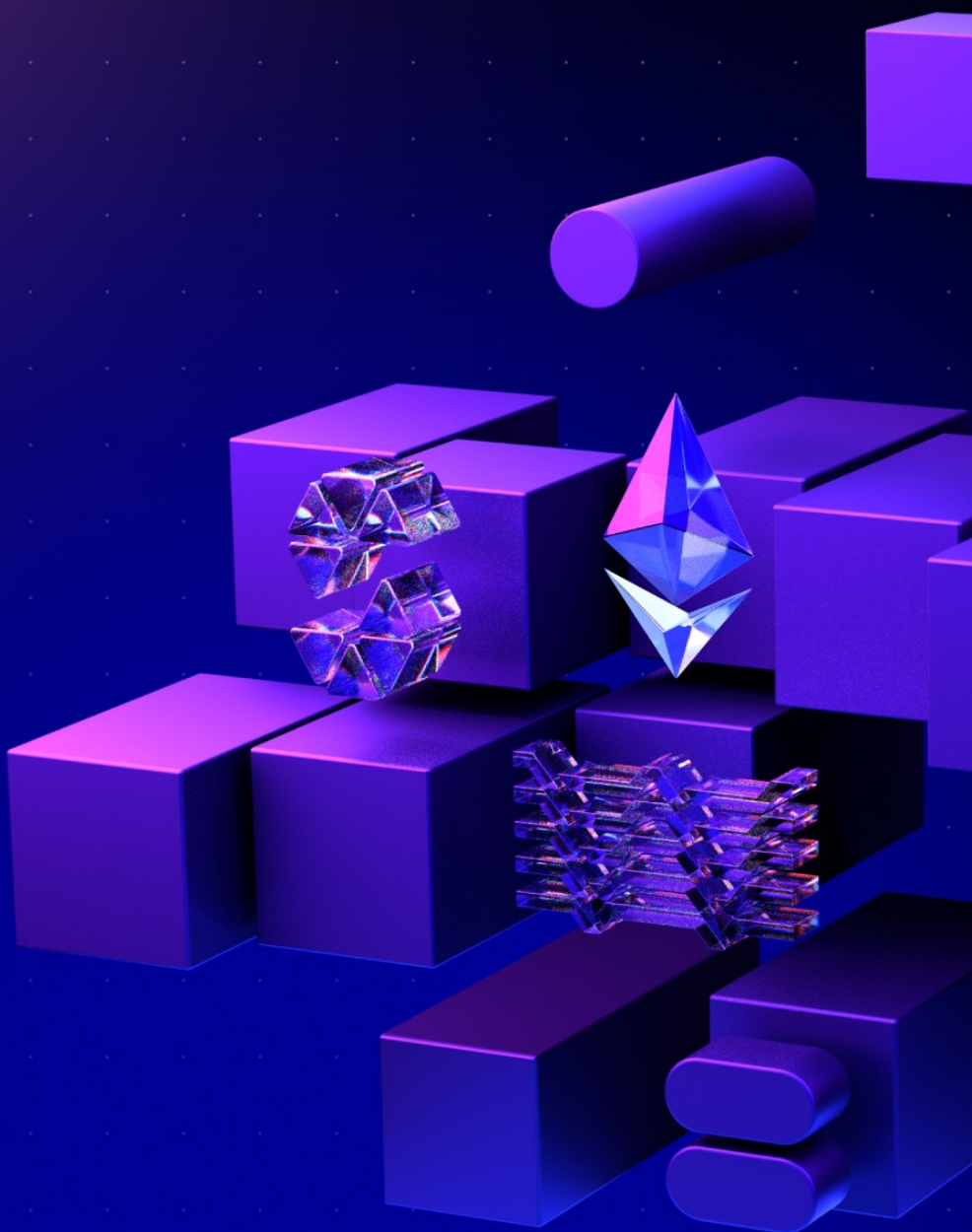
ackee
blockchain security

Greenhood

Contracts

18.8.2025



Audit Report

Contents

1. Document Revisions

1.0-draft	Draft Report	08.08.2025
1.0	Final Report	13.08.2025
1.1	Final Report	18.08.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.
Rohanske nabrezi 717/4
186 00 Prague, Czech Republic
https://ackee.xyz
hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**
   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**
   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**
   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**
   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**
   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Martin Veselý	Lead Auditor
Štěpán Šonský	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Greenhood is a protocol that enables regulated security token investments through a membership-based system. Users subscribe to obtain membership, which grants them a soulbound NFT and security token rewards. After becoming members, users can purchase additional security tokens. The system leverages T-REX (Token for Regulated EXchanges) infrastructure for regulatory compliance and implements role-based access controls for secure operation.

Revision 1.0

Greenhood engaged Ackee Blockchain Security to perform a security review of Greenhood Contracts with a total time donation of 3 engineering days in a period between August 4 and August 8, 2025, with Martin Veselý as the lead auditor.

The audit was performed on the commit `b12392f`[1] in the contracts repository and the scope was the following:

- `src/GreenhoodMembership.sol`; and
- `src/GreenhoodInvestor.sol`.

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved Wake testing framework. During the review, we paid special attention to:

- ensuring the arithmetic of the system is correct;
- detecting possible reentrancies in the code;
- ensuring access controls are not too relaxed or too strict; and
- looking for common issues such as data validation.

Our review resulted in 5 findings, ranging from Warning to High severity. The most severe finding H1 identified missing `whenNotPaused` modifiers in subscription functions, which could allow continued protocol interaction during emergency pauses.

Ackee Blockchain Security recommends Greenhood:

- implement timelocks or limits for critical parameter changes (exchange rates, subscription fees, reward amounts) to enhance user trust;
- add slippage protection in token purchase functions to prevent front-running;
- add zero address and zero amount validation checks in all relevant functions;
- review and enhance the pause mechanism implementation; and
- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Greenhood engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

Greenhood provided a pull request with the fixes. The changes significantly strengthened the protocol's trust model by implementing permissionless governance mechanisms and enhanced user protections.

The review was performed between August 13 and August 14, 2025 on the commit `9fd11a2`[2].

From the reported 5 findings:

- 5 issues were fixed;

- 0 issues were acknowledged;
- 0 minor issues were fixed partially; and
- 0 minor issues remained unresolved.

No new findings were reported.

[1] full commit hash: `b12392f36442ea5e558ef8d671c8380aca9f5e54`

[2] full commit hash: `9fd11a2c90ae1dd9e96ca586e4f916079b95d1c5`

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	1	2	0	2	0	5

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
H1: Missing `whenNotPaused` Modifiers in Subscription Functions	High	1.0	Fixed
M1: Parameter Front-running Possible Due to Instant Changes of Rates, Fees and Rewards	Medium	1.0	Fixed
M2: Unlimited subscriptionFee	Medium	1.0	Fixed

Finding title	Severity	Reported	Status
W1: Missing Zero Address and Zero Amount Validation Checks	Warning	1.0	Fixed
W2: One-step ownership transfer	Warning	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Martin Veselý	Lead Auditor
Štěpán Šonský	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The Greenhood protocol is a security token investment platform that combines membership benefits with investment opportunities. Users become members by paying a subscription fee, receiving both a soulbound NFT membership token and security token rewards. Members can then purchase additional security tokens through an investment gateway at configurable exchange rates.

The protocol leverages T-REX (Token for Regulated EXchanges) for regulatory compliance and implements gasless transactions through ERC20 permit functionality, reducing friction in the investment process. The system's components feature pausable functionality and role-based access controls, enabling emergency stops and privileged operations by designated administrators.

Trust Model

The protocol places significant trust in the owner role, which has extensive control over critical parameters:

- instant modification of exchange rates without limits;
- unrestricted adjustment of subscription fees and reward amounts; and

- no timelock mechanisms for parameter changes.

The membership NFT minting is controlled by token agents (primarily the membership contract), while the T-REX token's compliance features are treated as a trusted external component. These broad administrative powers, especially around instant parameter modifications without protection mechanisms, present notable centralization risks that are addressed in specific findings.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

H1: Missing `whenNotPaused` Modifiers in Subscription Functions

High severity issue

Impact:	Medium	Likelihood:	High
Target:	GreenhoodMembership.sol	Type:	Code quality

Description

Contract `GreenhoodMembership` implements `Pausable` functionality but does not use it effectively. The contract inherits from OpenZeppelin's `Pausable` contract and provides `pause` and `unpause` functions, indicating that subscription functionality should be pausable. However, the critical subscription functions `subscribe` and `subscribeWithPermit` lack the `whenNotPaused` modifier.

This renders the pause mechanism non-functional, preventing the owner from halting subscription operations during emergency situations.

Exploit scenario

1. Alice, the protocol owner decides to disable new subsriptions;
2. Alice immediately calls `pause` to prevent any further interactions with the protocol while the team prepares a fix;
3. Bob, a malicious actor who noticed the vulnerability, realizes that the `subscribe` and `subscribeWithPermit` functions are still accessible;
4. Despite the protocol being paused, Bob can successfully call these functions to subscribe and receive security tokens; and
5. The pause mechanism, intended as an emergency brake, fails to protect the protocol as Bob and other users can continue to interact with the vulnerable system.

The missing `whenNotPaused` modifier makes the emergency pause functionality ineffective for membership operations, undermining the protocol's ability to handle security incidents.

Recommendation

Add the `whenNotPaused` modifier to the `subscribe` and `subscribeWithPermit` functions.

Fix 1.1

The client fixed the issue by adding the `whenNotPaused` modifier to both subscription functions: `subscribe` and `subscribeWithPermit`.

Go back to Findings Summary

M1: Parameter Front-running Possible Due to Instant Changes of Rates, Fees and Rewards

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	GreenhoodInvestor.sol, GreenhoodMembership.sol	Type:	Front-running

Description

The protocol's purchase and subscription functions are vulnerable to parameter manipulation:

Purchase Functions: Both `purchaseTokens` and `purchaseTokensWithPermit` functions are vulnerable to exchange rate manipulation. While the amount of payment tokens is fixed (either by approval or permit), the owner can front-run the transaction by changing the exchange rate, causing users to receive fewer security tokens than expected. There is no protection against this as the exchange rate is only checked at execution time.

Subscribe Functions: The `subscribe` function is vulnerable to subscription fee front-running if users have approved more tokens than the current fee (a common practice). The `subscribeWithPermit` function is protected against fee changes as it requires an exact amount match in the permit. However, both functions are vulnerable to reward amount manipulation - the owner can decrease the reward amount just before the transaction executes, causing users to receive fewer security tokens than expected.

Exploit scenario

Alice, a regular user, wants to interact with the protocol under these conditions:

- investment: exchange rate is 1:1 (1000 payment tokens = 1000 security tokens); and
- subscription: fee is 1000 tokens, reward is 1000 security tokens.

Alice submits two transactions:

1. a purchase transaction: `purchaseTokensWithPermit` with permit for exactly 1000 tokens; and
2. a subscription transaction: `subscribeWithPermit` with permit for exactly 1000 tokens.

Bob, the malicious protocol owner, front-runs Alice's transactions by:

1. increasing the exchange rate to 2:1; and
2. decreasing the subscription reward to 500 tokens.

As a result:

- for the purchase: Alice pays 1000 tokens but receives only 500 security tokens; and
- for the subscription: Alice pays the expected 1000 tokens but receives only 500 reward tokens.

Recommendation

- implement timelocks for all critical parameter changes; and
- add slippage protection by introducing `minSecurityTokensToReceive` parameter to both purchase and subscription functions to ensure users receive their expected amount of tokens;

Fix 1.1

The client fixed the issue by:

- adding the `minTokensRequested` parameter to `purchaseTokens` and `purchaseTokensWithPermit` functions, and the `minRewardRequested` parameter to `subscribe` and `subscribeWithPermit` functions; and
- using a timelock contract as the owner of the protocol.

Go back to Findings Summary

M2: Unlimited subscriptionFee

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	GreenhoodMembership.sol	Type:	Data validation

Description

The owner is able to set unlimited `subscriptionFee` parameter.

Exploit scenario

1. Alice wants to subscribe to the protocol.
2. Alice accidentally approves the contract to spend an unlimited amount of payment tokens using a malicious front-end.
3. Bob, the protocol owner, sets `subscriptionFee` to the user's balance.
4. Alice calls the `subscribe` function.
5. Alice's wallet is drained.

Recommendation

- Implement immutable upper limit for `subscriptionFee` parameter.
- Add a timelock for `subscriptionFee` parameter.

Fix 1.1

The client fixed the issue by implementing an upper limit for the `subscriptionFee` parameter in the `setSubscriptionFee` function.

Go back to Findings Summary

W1: Missing Zero Address and Zero Amount Validation Checks

Impact:	Warning	Likelihood:	N/A
Target:	GreenhoodInvestor.sol, GreenhoodMembership.sol	Type:	Data validation

Description

There are missing checks for either zero addresses or zero amounts in the following functions:

GreenhoodMembership:

- `constructor`
- `setRewardAmount(uint256 rewardAmount_)`
- `setSubscriptionFee(uint256 subscriptionFee_)`
- `withdrawFunds(address to, uint256 amount)`

GreenhoodInvestor:

- `constructor`
- `setExchangeRate(uint256 exchangeRate_)`
- `withdrawFunds(address to, uint256 amount)`

Recommendation

- add zero address validation checks in constructors and functions accepting addresses; and
- add zero amount validation checks for all functions accepting numerical parameters;

Fix 1.1

The client fixed the issue by adding validation checks for zero addresses and zero amounts in all affected functions.

Go back to Findings Summary

W2: One-step ownership transfer

Impact:	Warning	Likelihood:	N/A
Target:	GreenhoodInvestor.sol, GreenhoodMembership.sol	Type:	Access control

Description

Contracts `GreenhoodMembership` and `GreenhoodInvestor` use OpenZeppelin `Ownable` with one-step ownership transfer which can lead to accidental loss of ownership.

Recommendation

Use `Ownable2Step` instead of `Ownable`.

Fix 1.1

The client fixed the issue by using `Ownable2Step` instead of `Ownable`.

Go back to Findings Summary

Report Revision 1.1

Revision Team

Revision team is the same as in Report Revision 1.0.

System Overview

The Greenhood protocol continues to operate as a security token investment platform where users become members through subscription, receiving both a soulbound NFT membership token and security token rewards. Members can purchase additional security tokens through an investment gateway at configurable exchange rates.

The system maintains its integration with T-REX for regulatory compliance and ERC20 permit functionality for gasless transactions. New safety mechanisms include timelock-controlled administrative operations, capped subscription fees, and user-specified minimum values for rewards and purchased tokens. These enhancements strengthen the platform's security while preserving its core investment and membership features.

Trust Model

The protocol has strengthened its security model through enhanced controls and user protections:

Administrative Controls:

- the owner role operates through a timelock contract, ensuring transparency for all parameter changes;
- subscription fees cannot exceed a fixed maximum value; and
- parameter changes require a waiting period before taking effect.

User Protections:

- subscription functions accept minimum reward parameters to prevent front-running;
- purchase functions include slippage protection through minimum token parameters; and
- all critical parameter changes are visible on-chain before execution.

These improvements maintain protocol flexibility while providing robust safeguards against parameter manipulation.

Appendix A: How to cite

Please cite this document as:

> Ackee Blockchain Security, Audit Report | Greenhood: Contracts, 18.8.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Detectors

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz