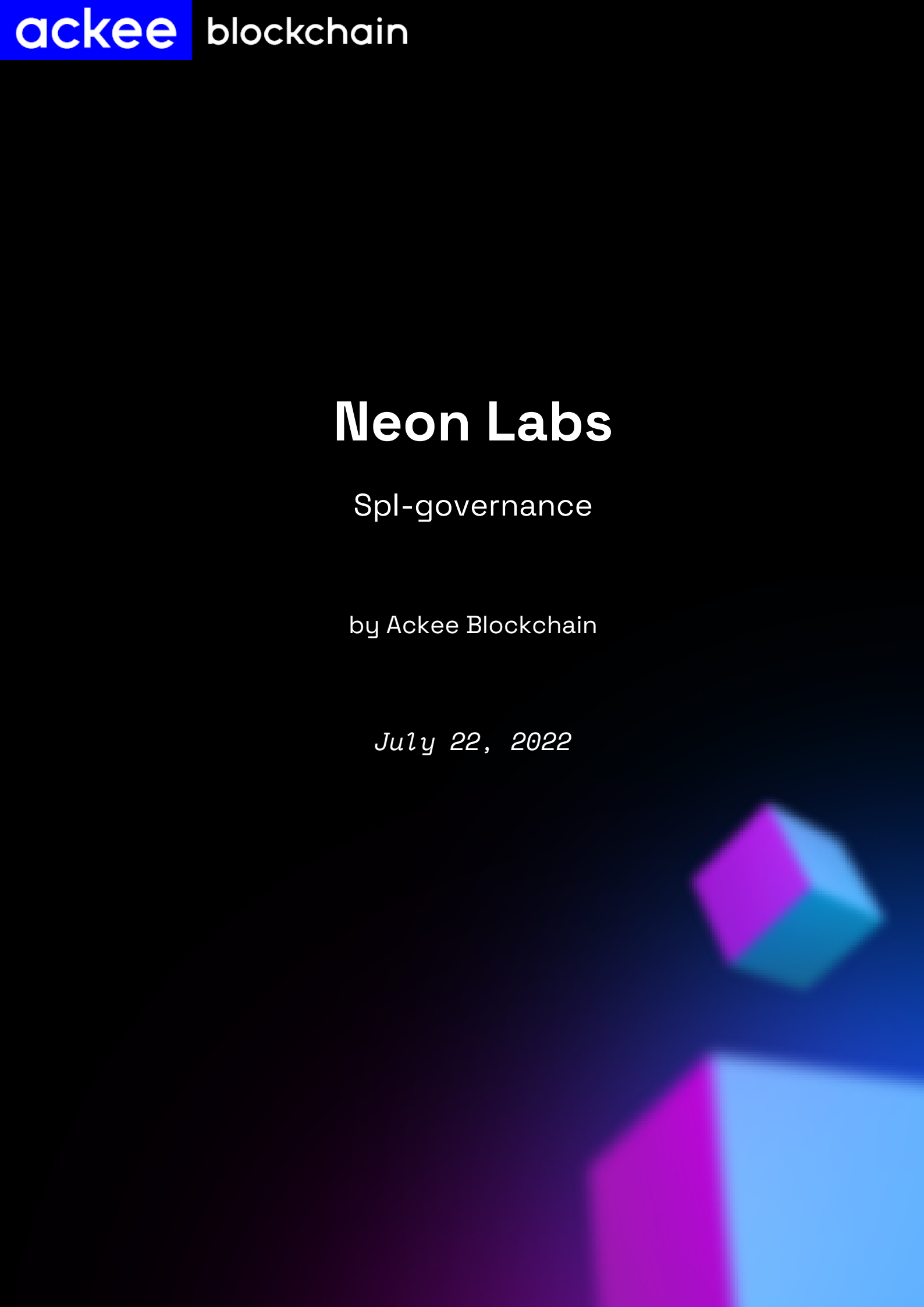
ackee blockchain

Neon Labs

Spl-governance

by Ackee Blockchain

July 22, 2022



Contents

1. Document Revisions

0.1	Critical issue report, draft-report	July 15, 2022
1.0	Final report	July 22, 2022
1.1	Fix Review	Sep 5, 2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2. Audit Methodology

The Ackee Blockchain auditing process follows a routine series of steps:

1. **Code review**
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

    missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows, ...

c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.

d. Review of best practices to improve efficiency, clarity, and maintainability.

2. **Testing and automated analysis**

   a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trdelnik.

3. **Local deployment + hacking**

   a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are characteristic of each program audited. However, when trying to attack, we rely on the information gained from previous steps and our rich experience.

2.3. Review team

Member's Name	Position
Vladimír Marcin	Lead Auditor
Tibor Tribus	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We have put our best effort into finding all vulnerabilities in the system; however, our findings should not 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

Neon Labs engaged Ackee Blockchain to conduct a security review of their SPL Governance contract with a total time donation of 26 engineering days. The review took place between 27. June and 22. July 2022.

The scope included the following repository with a given commit:

- **Repository**: spl-governance
- **Commit**: `f13d7e7c1507819306797688ce0bb1f6950a5038`
- **Programs**: `maintenance/program`, `addin-fixed-weights/program`, `addin-vesting/program`, `governance-lib`

The beginning of the audit was dedicated to understanding the Neon governance. We then took a deep dive into the `maintenance` program and custom Neon addins: `addin-fixed-weights/program`, `addin-vesting/program`. During the review, we paid particular attention to:

- Is the correctness of the custom addins ensured (do they correctly implement spl-governance contract specification)?
- Do the program correctly use dependencies or other programs they rely on (e.g., SPL dependencies)?
- Is the code vulnerable to voting manipulation?

Our review resulted in 2 *`Critical`* severity issues and another six findings ranging from *`Informational`* to *`Medium`* severity. In total, we identified eight findings. The most severe one would allow an attacker to increase the weight of his/her vote to such an extent that he/she can practically decide on any proposal by himself/herself, and it was immediately reported to the client (see C1: Possibility to manipulate a voting process while using the fixed-weights addin).

AckeeBlockchain recommends Neon Labs:

- address all reported issues,
- monitor the SPL governance program and apply major changes in the future, as the program is still in active development.

Update Sep 5, 2022: Neon Labs provided an updated codebase that addresses issues from this report. See Appendix A for a detailed discussion of the exact status of each issue.

4. System Overview

This section contains an outline of the audited programs. Note that this is meant for understandability purposes and does not replace project documentation.

4.1. Programs

Programs we find important for better understanding are described in the following section.

SPL governance

The two following programs are addins to the SPL Governance program. For more information, see the official SPL Governance docs.

Addin-fixed-weights

The addin is used to create decentralized management until the moment of token issuance.

Addin-vesting

Implements functionality similar to the basic deposit/withdraw functionality inside the spl-governance program to connect a user to voting capabilities. In addition to this, implements:

1. The ability to make a schedule for the withdrawal of the locked tokens in whole or in parts at specific points in time.
2. The ability to vote by a part of the user's locked tokens.

Maintanance program

The `maintenance` program is a solana on-chain program responsible for delegating maintenance over a maintained program. The process of upgrade

can be triggered by one of the delegates. The number of delegates cannot be more than ten.

Actors

- **Delegate** - A delegate can trigger the process of upgrading the maintained program but is limited only to approved code hashes.
- **Authority** - An authority has the right to set delegates and code hashes for future program upgrades.

Trust model

A user of a maintained program must trust the maintenance authority as it decides what code will be whitelisted.

5. Vulnerabilities risk methodology

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 (such as deployment scripts, compiler configuration, use of 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.

5.1. Finding classification

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	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 (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), 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 (see above) 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.

6. Findings

This section contains the list of discovered findings. Unless overriden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Severity	Impact	Likelihood
C1: Possibility to manipulate a voting process while using the fixed-weights addin	Critical	High	High
C2: When using the addin-vesting (for realm), the first user will be able to decide on any proposal after his deposit	Critical	High	High
M1: Possibility to decide on a proposal without a sufficient voting weight	Medium	High	Low

	Severity	Impact	Likelihood
M2: Possibility of a DoS attack that prevents the creation of a valid maintenance record	Medium	High	Low
L1: Using `find_program_address` instead of `create_program_address`	Low	Low	Low
I1: Unused account	Info	Info	N/A
I2: Misleading docs	Info	Info	N/A
I3: Hanging accounts	Info	Info	N/A

Table 2. Table of Findings

C1: Possibility to manipulate a voting process while using the fixed-weights addin

Critical severity issue

Impact:	High	Likelihood:	High
Target:	SetVotePercentage	Type:	Data validation

Description

In the `SetVotePercentage` instruction of the `addin-fixed-weights` program, a condition that would limit the `vote_percentage` is missing. The consequence is that a caller can send a number greater than 10.000 as a parameter determining the vote percentage, which can increase the power of the vote several times (note: a number greater than 10.000 means more than 100%).

It will then allow one of the users (members of governance) to increase the weight of his/her vote to such an extent that he/she can practically decide on any proposal by himself/herself.

Exploit scenario

1. An attacker is specified in the `VOTER_LIST`.
2. The attacker creates a proposal that will be advantageous for him. So no one but him would vote for such a proposal. However, the attacker increases the weight of his vote to such a level that he will approve this proposal himself.
3. Moreover, since switching between addins is also controlled by the DAO, our attacker can effectively forever block the proposal that will propose this switch.

Recommendation

Add the following condition to the `SetVotePercentage` instruction of the `addin-fixed-weights` program:

```
if vote_percentage > 10000 {
    return Err(VoterWeightAddinError::InvalidPercentage.into());
}
```

Go back to Findings Summary

C2: When using the addin-vesting (for realm), the first user will be able to decide on any proposal after his deposit

Critical severity issue

Impact:	High	Likelihood:	High
Target:	Deposit	Type:	Program logic

Description

In the `Deposit` instruction (for realm) from the `addin-vesting` program, a `voter_weight` record is created for each user (future governance member), and the weight of his/her vote is derived from the number of deposited tokens (`total_amount`). However, in the case of the first deposit, when there is no `max_voter_weight_record` yet, this record is also created by the `Deposit` instruction, and its weight is equal to the `total_amount` of the first deposit (each subsequent deposit adds its `total_amount` to the weight of `max_voter_weight` record).

It follows from the above that the vote of the first governance member who will use this addin will have a weight of 100% and thus will be able to approve any proposal.

Exploit scenario

1. The first user who calls a deposit can immediately create a proposal and vote for it. Since the weight of his vote is 100%, the given proposal will automatically be marked as successful (early tipping must be enabled).

Recommendation

It must be ensured that the switch of addins can only happen in a situation

when all members (or at least a ”sufficient” number of them) have made a deposit and thus, the `max_voter_weight` record is sufficiently scaled.

Go back to Findings Summary

M1: Possibility to decide on a proposal without a sufficient voting weight

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	-	Type:	Addin switch

Description

If there are still some active proposals (they are in the `Voting` state) when switching from `fixed-weight-addin` to `vesting-addin`, there may be a situation when these proposals become successful due to the addin switch. The reason is that when switching, the `max_voter_weight` changes (therefore, the percentage ratio of votes collected so far will also change), and thus the collected votes can suddenly become sufficient for the given proposal to be accepted even if it would have been rejected when using the old addin.

Recommendation

It should be ensured that there are no active proposals during the switching of addins.

Go back to Findings Summary

M2: Possibility of a DoS attack that prevents the creation of a valid maintenance record

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	CreateMaintenance	Type:	DoS

Description

The problem is that anyone can call the `create_maintenance` instruction. This instruction creates a maintenance record whose address is the PDA derived from the address of the maintained program. An attacker can thus call this instruction and create a maintenance record for a foreign program, but he/she will appear as a maintenance authority in this newly created record.

It prevents the current upgrade authority of the program from using the address of the maintenance record as the new upgrade authority, as it would lead to the situation where the attacker would gain control over the upgrade of the program. The only option would be to redeploy the program, which will change its public key, and thus it will be possible to create another maintenance record.

Exploit scenario

1. The attacker somehow identifies the `program_id` that the maintenance program should maintain.
2. Attacker calls the `create_maintenance` instruction to which he sent this identified key and simultaneously sets himself as maintenance authority.
3. The attacker will effectively block the address of the maintenance record for the given `program_id`, and thus no one else will be able to create a maintenance record for the given program.

Recommendation

Only the current upgrade authority of a program that will be maintained should be able to call the `create_maintenance` instruction. As part of this instruction, the `UpgradeableLoaderInstruction::SetAuthority` would be called, and it sets the address of the newly created maintenance record as the new upgrade authority.

Go back to Findings Summary

L1: Using `find_program_address` instead of `create_program_address`

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	-	Type:	Compute budget

Description

By calling the `find_program_address` function, there is a risk of exceeding the instruction compute budget. The process of finding a valid program address is by trial and error, and even though it is deterministic given a set of inputs, it can take a variable amount of time to succeed across different inputs. When called from an on-chain program, it may incur a variable amount of the program's compute budget.

Recommendation

During the creation and initialization of `Vesting` and `Maintenance` records, the function `find_program_address` is called. The bump calculated by the `find_program_address` function could be stored in these records, and then it could be used by the `create_program_address` function when it is necessary to recalculate the addresses of these accounts.

Go back to Findings Summary

I1: Unused account

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Account management

Description

The `Deposit` instruction accepts the `governance_program` account as the input but never calls any instruction of this program. It makes this account useless in this context and should be removed as it unnecessarily increases the size of the transaction.

The same recommendation applies to the `CreateVoterWeightRecord` instruction.

Recommendation

Explain or delete this account. By removing an unused account, one can effectively reduce the size of a transaction.

Go back to Findings Summary

I2: Misleading docs

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Code smell

Description

Misleading documentation inside the maintenance program. (LoC: 17).
Probably a copy-paste mistake.

Recommendation

Adjust documentation to match the code.

Go back to Findings Summary

I3: Hanging accounts

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Accounts management

Description

There is currently no way to close (and by that, redeem the fees from) `VoterWeightRecord` accounts. The fees (lamports paid for the account creation) stay there forever.

Recommendation

Add the functionality to close these accounts. You can get some fees back, and at the same time, there will be no unnecessary/unused accounts on the chain.

Go back to Findings Summary

Appendix A: Fix Review

On Sep 5, 2022, Ackee Blockchain reviewed Neon Labs's fixes for the issues identified in this report. The following table summarizes the fix review. The updated commit was `be99feed8d0143ad3a77f9f50e25c97015ace3d2`.

Fix log

Id	Severity	Impact	Likelihood	Status
C1F: Possibility to manipulate a voting process while using the fixed-weights addin	Critical	High	High	**Fixed**
C2F: When using the addin-vesting (for realm), the first user will be able to decide on any proposal after his deposit	Critical	High	High	**Acknowledged**
M1F: Possibility to decide on a proposal without a sufficient voting weight	Medium	High	Low	**Acknowledged**
M2F: Possibility of a DoS attack that prevents the creation of a valid maintenance record	Medium	High	Low	**Fixed**

Id	Severity	Impact	Likelihood	Status
L1F: Using `find_program_address` instead of `create_program_address`	Low	Low	Low	**Acknowledged**
I1F: Unused account	Info	Info	N/A	**Fixed**
I2F: Misleading docs	Info	Info	N/A	**Fixed**
I3F: Hanging accounts	Info	Info	N/A	**Partially fixed**

Table 3. Table of fixes

C1F: Possibility to manipulate a voting process while using the fixed-weights addin

Critical severity issue

Impact:	High	Likelihood:	High
Target:	C1: Possibility to manipulate a voting process while using the fixed-weights addin	Type:	Data validation

Description

The exploit scenario from the C1 issue is no longer possible. The following condition was added to the `SetVotePercentage` instruction of the `addin-fixed-weights` program:

```
if vote_percentage > 10000 {
    return Err(VoterWeightAddinError::InvalidPercentage.into());
}
```

Go back to Fix log

C2F: When using the addin-vesting (for realm), the first user will be able to decide on any proposal after his deposit

Critical severity issue

Impact:	High	Likelihood:	High
Target:	C2: When using the addin-vesting (for realm), the first user will be able to decide on any proposal after his deposit	Type:	Program logic

Description

The issue is handled in the off-chain startup scripts. The switch of addins happens after the token issuance, and before the switch, tokens will be distributed according to weights given by the `addin-fixed-weights`.

The distribution is performed by calling the `Deposit` instruction from the `addin-vesting` program on behalf of individual voters. It will ensure that the `max_voter_weight` record of `addin-vesting` will have the same value as the previously used `max_voter_weight` of the `addin-fixed-weights` program.

Using the described approach to switch addins, the exploit scenario from the C2 issue is not possible. For more details, see the proposal_tge.rs file.

Warning: As the source code of the neon spl-governance is open source and free to use, it is essential to note that the exploit scenario is still possible when using the addin-vesting without the described token distribution.

Go back to Fix log

M1F: Possibility to decide on a proposal without a sufficient voting weight

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	M1: Possibility to decide on a proposal without a sufficient voting weight	Type:	Addin switch

Description

The off-chain startup scripts also handle the M1 issue. The token distribution described in the C2F fix ensures that the situation described in the M1 issue will never happen.

Go back to Fix log

M2F: Possibility of a DoS attack that prevents the creation of a valid maintenance record

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	M2: Possibility of a DoS attack that prevents the creation of a valid maintenance record	Type:	DoS

Description

The issue was fixed according to our recommendation. Only the current upgrade authority of a maintained program can call the `create_maintanance` instruction, which makes it impossible for an attacker to perform the DoS attack.

Go back to Fix log

L1F: Using `find_program_address` instead of `create_program_address`

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	L1: Using `find_program_address` instead of `create_program_address`	Type:	Compute budget

Description

The Neon team acknowledged the issue. However, the low severity and the complexity of the fix were good enough reasons not to fix this finding, as the fix would require redesigning the whole storage schema.

Go back to Fix log

I1F: Unused account

Impact:	Info	Likelihood:	N/A
Target:	I1: Unused account	Type:	Account management

Description

The unused account was removed.

Go back to Fix log

I2F: Misleading docs

Impact:	Info	Likelihood:	N/A
Target:	I2: Misleading docs	Type:	Code smell

Description

The documentation now matches the code.

Go back to Fix log

I3F: Hanging accounts

Impact:	Info	Likelihood:	N/A
Target:	I3: Hanging accounts	Type:	Accounts management

Description

The Neon team has added functionality that ensures the closing of `VestingRecord` and `VoterWeightRecord` accounts in the `addin-vesting` program.

The team stated that there is no need to close the `VoterWeightRecord` accounts in the `adding-fixed-wights` due to the small number of these accounts.

Go back to Fix log

Appendix B: How to cite

Please cite this document as:

Ackee Blockchain, Neon Labs: Spl-governance, July 22, 2022.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq