Zellic

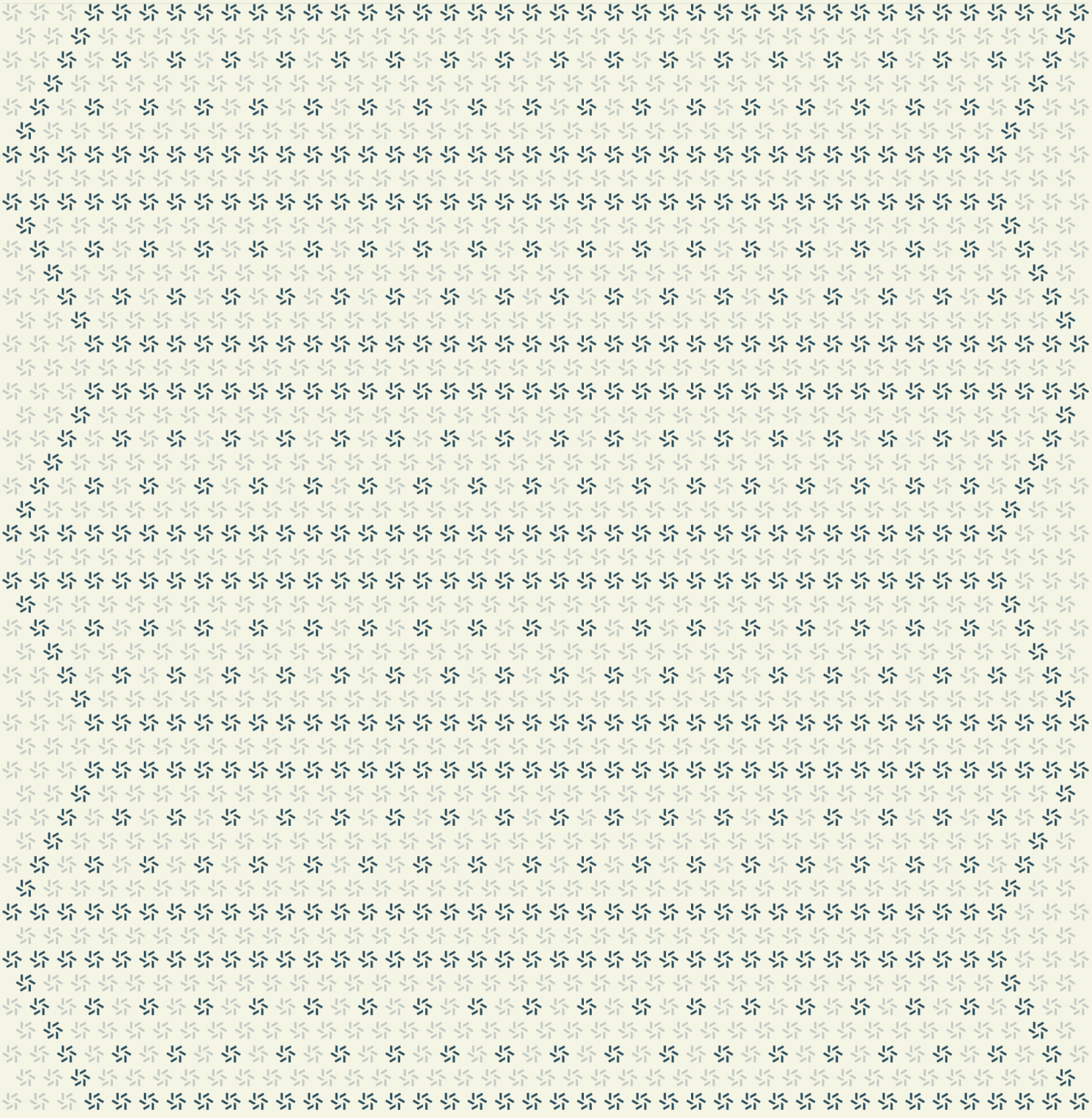
Prepared for
Will Hickey
Anza

Prepared by
Frank Bachman
Junyi Wang
Zellic

October 29, 2024

Solana core BPF programs

Solana Application Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Anza from September 17, to September 26, 2024. During this engagement, Zellic reviewed Solana core BPF programs's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any usual on-chain issues such as missing signer or owner checks?
- Are there any differences in behavior compared to the native programs?
- Are the data-management operations implemented correctly in the address-lookup-table program?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Solana core BPF programs contracts, there were no security vulnerabilities discovered.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	0

2. Introduction

2.1. About Solana core BPF programs

Anza contributed the following description of Solana core BPF programs:

> These are the core BPF programs that will replace the native programs on Solana.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and

Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Solana core BPF programs Contracts

Type	Rust
Platform	Solana
Target	address-lookup-table
Repository	https://github.com/solana-program/address-lookup-table ↗
Version	`af6ed4078b7ffc596b77033d8ae67dfeef343cf6`
Programs	`address-lookup-table`
Target	config
Repository	https://github.com/solana-program/config ↗
Version	`eff3b5a03348782572ec0d058f3f6ec033724635`
Programs	`config`

Target	feature-gate
Repository	https://github.com/solana-program/feature-gate ↗
Version	`0f12ca891aa6bd36b43caf9c8f412067954de926`
Programs	`feature-gate`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of three person-weeks. The assessment was conducted by two consultants over the course of two calendar week.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Frank Bachman
Engineer
frank@zellic.io ↗

Junyi Wang
Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 17, 2024	Start of primary review period
Septemer 26, 2024	End of primary review period

3. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

3.1. Module: feature-gate

Function: `process_revoke_pending_activation(_program_id, accounts)`

This revokes a pending activation by feature-info accounts and burning the lamports stored there.

Inputs

- `feature_info`
  - **Validation**: Must be a signer and owned by the current program.
  - **Impact**: This is the account that is destroyed.
- `incinerator_info`
  - **Validation**: None.
  - **Impact**: The excess lamports are sent here.

Branches and code coverage (including function calls)

Intended branches

- Destroys the feature-info account properly.
  - ☑ Test coverage

Negative behavior

- Reverts if `feature_info` is not a signer.
  - ☑ Negative test
- Reverts if `feature_info` is not owned by the current program.
  - ☐ Negative test
- Reverts if the feature is already activated.
  - ☑ Negative test

3.2. Module: config

Function: `process(program_id, accounts, input)`

This processes an instruction updating the config stored at an account. It checks signatures to ensure the update is authorized.

Inputs

- `config_account`
  - **Validation**: Must be owned by the current program.
  - **Impact**: This is the config account that is updated.
- `...signers`
  - **Validation**: Must all be signers.
  - **Impact**: These signers are used to validate the request.
- `input`
  - **Validation**: The input correctly deserializes into a `ConfigKeys` structure.
  - **Impact**: The config account is updated to contain the input.

Branches and code coverage (including function calls)

Intended branches

- Updates the config account correctly.
  - ☑ Test coverage

Negative behavior

- Reverts if too few signers are present.
  - ☑ Negative test
- Reverts if the new data exceeds the space available in the config account.
  - ☑ Negative test
- Reverts if the new keys contain duplicates.
  - ☑ Negative test
- Reverts if the address provided after the config account is not a signer.
  - ☑ Negative test
- Reverts if the address provided after the config account does not match the key in `input` in the corresponding position.
  - ☐ Negative test
- Reverts if the list of signers is empty and the config account is not a signer.
  - ☑ Negative test
- Reverts if a purported signer is not a signer in the config account.
  - ☑ Negative test
- Reverts if the config account is not owned by the current program.

- ☑ Negative test

3.3. Module: address-lookup-table

Function: `process_close_lookup_table(program_id, accounts)`

This closes a deactivated lookup table and returns the rent lamports.

Inputs

- `lookup_table_info`
  - **Validation**: Must be owned by the current program.
  - **Impact**: This is the lookup table that is closed.
- `authority_info`
  - **Validation**: Must be a signer.
  - **Impact**: This is used to verify that the caller is authorized to close the table.
- `recipient_info`
  - **Validation**: None, not necessary as it only receives funds.
  - **Impact**: The receipts of the refunded lamports from closing the table.

Branches and code coverage (including function calls)

Intended branches

- Closes the table and refunds the lamports to the recipient on the happy path.
  - ☑ Test coverage

Negative behavior

- Reverts if the table is not owned by the current program.
  - ☐ Negative test
- Reverts if the authority given is not a signer.
  - ☑ Negative test
- Reverts if the table is frozen.
  - ☐ Negative test
- Reverts if the authority given does not match the authority of the lookup table.
  - ☑ Negative test
- Reverts if the table is not fully deactivated(in the deactivated state for a certain number of blocks).
  - ☑ Negative test

Function: `process_create_lookup_table(program_id, accounts, untrusted_recent_slot, bump_seed)`

This creates a new account for the address-lookup-table and initializes the metadata on it. The `relax_authority_signer_check_for_lookup_table_creation` feature is now enabled on all clusters, and hence the relevant checks present on the native program have been removed.

Inputs

- `untrusted_recent_slot`
  - **Validation**: Used to fetch a slot hash, which only succeeds if it is a valid slot.
  - **Impact**: Used to produce the slot hash, which derives the account address for the table.
- `bump_seed`
  - **Validation**: None.
  - **Impact**: Used in the derivation of the account address for the table.
- `lookup_table_info`
  - **Validation**: This account's key must match the key derived from the `authority_info`, the derivation slot, and the bump seed.
  - **Impact**: The account to contain the new lookup table.
- `authority_info`
  - **Validation**: This account's key is used to derive the `lookup_table_info`. It must match.
  - **Impact**: The account to contain the new lookup table.
- `payer_info`
  - **Validation**: Must be a signer.
  - **Impact**: This account pays the rent for the lookup table.

Branches and code coverage (including function calls)

Intended branches

- Stores a lookup table into the given account when it is not already created.
  - ☑ Test coverage
- Does nothing if the table is already created.
  - ☑ Test coverage
- Pays the rent using the given payer when the rent required is nonzero.
  - ☑ Test coverage

Negative behavior

- Reverts if the payer is not a signer.
  - ☐ Negative test
- Reverts if the given slot is invalid.

- ☑ Negative test
- Reverts if the derived table address does not match the given table address.
  - ☑ Negative test

Function: `process_deactivate_lookup_table(program_id, accounts)`

This deactivates the lookup table by setting the deactivation slot to the current slot.

Inputs

- `lookup_table_info`
  - **Validation**: Must be owned by the current program.
  - **Impact**: The lookup table that is deactivated.
- `authority_info`
  - **Validation**: Must be a signer.
  - **Impact**: Used to check if the caller is authorized to modify the current lookup table.

Branches and code coverage (including function calls)

Intended branches

- Deactivates the table on the happy path.
  - ☑ Test coverage

Negative behavior

- Reverts if the given lookup table is not owned by the current program.
  - ☐ Negative test
- Reverts if the authority is not a signer.
  - ☑ Negative test
- Reverts if the lookup table is frozen.
  - ☐ Negative test
- Reverts if the given authority does not match the authority of the lookup table.
  - ☑ Negative test
- Reverts if the table is already deactivated.
  - ☑ Negative test

Function: `process_extend_lookup_table(program_id, accounts, new_addresses)`

This extends the lookup table by adding more addresses to it at the end.

Inputs

- `lookup_table_info`
  - **Validation**: The account must be owned by the current program.
  - **Impact**: This is the table that is extended.
- `new_addresses`
  - **Validation**: None — but unnecessary.
  - **Impact**: These addresses are added to the end of the table.
- `authority_info`
  - **Validation**: Must be a signer.
  - **Impact**: This is used to check that the caller has the authority to call this function.
- `payer_info`
  - **Validation**: Must be a signer.
  - **Impact**: Lamports are transferred from this account to cover the rent increase caused by adding addresses.

Branches and code coverage (including function calls)

Intended branches

- Extends the table with the new addresses in the happy path.
  - ☑ Test coverage
- Transfers lamports to cover rent from the payer if more rent is needed.
  - ☑ Test coverage

Negative behavior

- Reverts if the payer is not a signer and payment is needed.
  - ☐ Negative test
- Reverts if the lookup table is not owned by the current program.
  - ☐ Negative test
- Reverts if the authority is not a signer.
  - ☑ Negative test
- Reverts if the lookup table is frozen.
  - ☑ Negative test
- Reverts if the lookup table is deactivated.
  - ☑ Negative test
- Reverts if the passed-in authority does not match the authority for the lookup table.
  - ☑ Negative test
- Reverts if the table is full.
  - ☑ Negative test
- Reverts if the list of new addresses is empty.
  - ☑ Negative test
- Reverts if extending the table would exceed the maximum capacity.

- ☑ Negative test

Function: `process_freeze_lookup_table(program_id, accounts)`

This freezes the lookup table by erasing the authority.

Inputs

- `lookup_table_info`
  - **Validation**: The current program owns the lookup table.
  - **Impact**: This is the lookup table that becomes frozen.
- `authority_info`
  - **Validation**: The account is a signer.
  - **Impact**: This is used to verify the caller is authorized. It checks that the `lookup_table` authority matches the provided account.

Branches and code coverage (including function calls)

Intended branches

- Overwrites the authority to be `None`.
  - ☑ Test coverage

Negative behavior

- Reverts if the authority passed in is not a signer.
  - ☑ Negative test
- Reverts if the lookup table is already frozen.
  - ☑ Negative test
- Reverts if the passed-in authority does not match that of the lookup table.
  - ☑ Negative test
- Reverts if the table is deactivated.
  - ☑ Negative test
- Reverts if the lookup table is empty.
  - ☑ Negative test

4. Assessment Results

At the time of our assessment, the reviewed code was not deployed to Solana Mainnet.

During our assessment on the scoped Solana core BPF programs contracts, there were no security vulnerabilities discovered.

4.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.