Code Review

Solana

21 September 2020
Version: 1.0

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision SA

Corporate Headquarters
Kudelski Security – Nagravision SA
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.0
File Name:	Solana Code Review Report - v1.docx
Publication Date:	21 September 2020
Confidentiality Level:	Confidential
Document Owner:	Kudelski Group – Research & Cryptography Team
Document Recipient:	Solana Technical Team
Document Status:	Draft

TABLE OF FIGURES

EXECUTIVE SUMMARY

Kudelski Security ("Kudelski"), the cybersecurity division of the Kudelski Group, was engaged by Solana Foundation. ("Solana") client to conduct an external security assessment in the form of a Code Review of the Solana Blockchain implementation.

The assessment was conducted remotely by the Kudelski Security Team from our secure lab environment. The tests took place from September 8, 2020 to September 18, 2020 and focused on the following objectives:

1. Derived addresses in the blockchain
2. Token Program implementation
3. Freeze Accounts capability

Following our analysis, we have manually reviewed and have verified the following:

- Review Solana's use of derived addresses to authorize withdrawals
    - Understand the increase in risk of using both derived addresses and public keys that can conceivably be treated as derived addresses by a malicious party
    - Investigate the potential for and impact of logic errors around the assumptions of how those sha256 (pubkey, seed) addresses are used
- Review Solana Token Program to create an updated baseline
    - https://github.com/solana-labs/solana-program-library/tree/master/token
- PR Review for Capability to Freeze Accounts
    - https://github.com/solana-labs/solana-program-library/pull/297

This report summarizes the tests performed and findings in terms of strengths and weaknesses. It also contains detailed descriptions of the discovered vulnerabilities, and recommendations for remediation.

1.1 Engagement Limitations

The project was time-boxed to be finished by September 18, 2020.

During the project the Kudelski teams has focused on the following areas

- Validate technical design claims and cryptographic coding underlying the behavior and intent of the technical systems
- Perform a code-review of provided Rust Code, especially focusing on code written by the internal team, assuming third-party libraries act as expected
- Validate implementation choices, completeness, and assumptions according to the design provisions and deployment
- Validate predictions and behaviors with special attention to decentralized and distributed behaviors for claims with special attention to the defined areas
- Provide recommendations for security related improvements and corrections to the infrastructure and architecture, if found
- Provide recommendations for architectural and implementation related improvements to the infrastructure and architecture, if found

Specifically, and as indicated by the client, the results will focus to answer the following questions:

- Understand the increase in risk of using derived addresses and public keys that can conceivably be treated as derived addresses by a malicious party
- Investigate the potential for and impacts of logic errors around the assumptions of how those sha256(pubkey, seed) addresses are used
- create an updated baseline of the Solana Token Program
- Capability to Freeze Accounts pull request review

Out of scope for this engagement, which can be included in future engagements include deployment of the infrastructure at-scale to validate findings, operational execution of the code to perform a pen-test of running binaries (memory review, attacks to binaries, theft of secrets), operational assessment of alerting and monitoring when non-ethical behavior is present in the system, or participation in any running test-net environments.

1.2 Engagement Analysis

The engagement was performed by cloning the relevant repositories into the internal GitLab environment utilized by Kudelski Security. The code base cloned was from the Master branch at 07 September, 2020 with the SHA 9eb10d914419fe448852f1aa1e6d65df9743c1e3 of the last commit. Considerable time and effort of the engagement was spent focusing on the facts deemed most relevant to the client as outlined in the initial document provided by SOLANA.

The code review was then conducted by analyzing the relevant code from different angles with both automated and manual tools.

As a result of our work, we identified **1 High**, **0 Medium**, **8 Low**, and **3 Informational** findings.

The high findings are either omissions & vulnerabilities in the code or insecure handling of cryptographic components, e.g. private key handling, number over/underflow in calculations and random number generation.

As we didn't find any medium findings there is nothing to comment.

Multiple low findings were identified during our assessment. Although these findings do not represent a significant threat, they can increase the attack surface of the system.

Several informational findings were found. These are mostly cosmetic or omissions in functionality in tools like the CLI client. These are well known and documented.

This report does not contain any re-verification that the above issues have been fixed within the codebase, but we expect to review the fixes in later assessments.

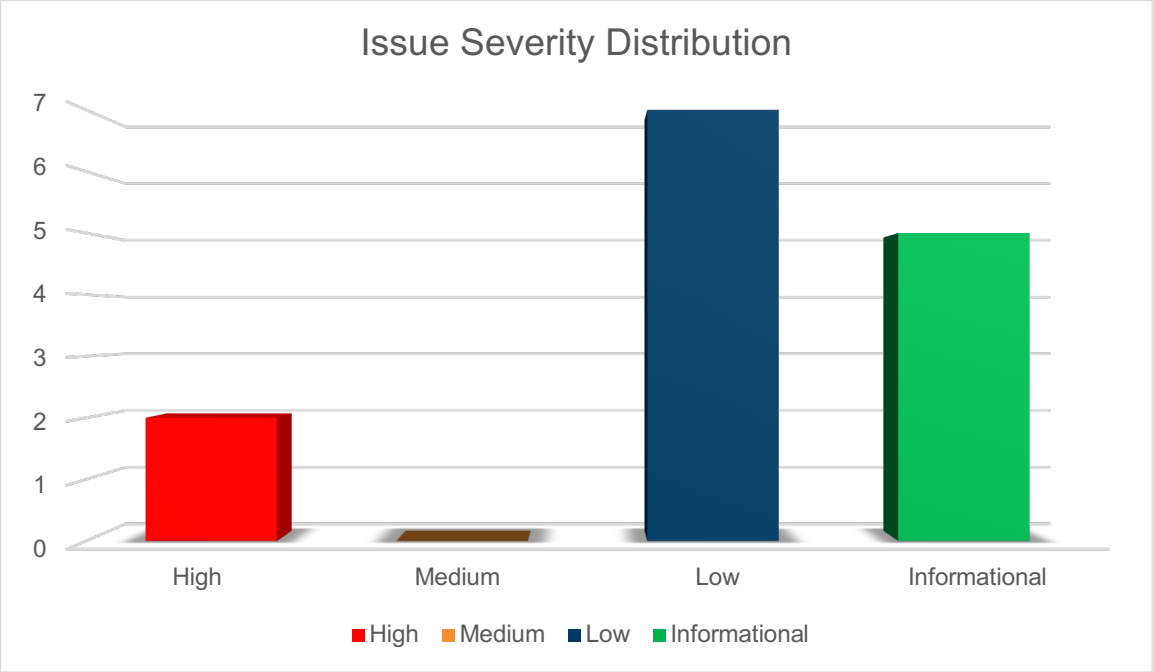

Figure 1 Issue Severity Distribution

1.3 Observations

1.3.1 Code and design documentation

We have structured the code review in three sections

- Derived addresses
- Token Program
- Freeze Accounts capability

We have performed the following analysis on security of the code

- Code construct and call-trees
- Verification that Consensus implementation in core code based on the architecture documentation provided.

Cryptographic analysis of the use of cryptographic primitives, both third-party libraries and self-developed implementation of the above-mentioned primitives.

The documentation is well written and covers the architecture and the implementation of it as code.

1.3.2 Coding Style

As this is a follow up on the previous code reviews, we must commend the development team on how much of the code quality that has improved. This has made much of the findings being on the bottom half of the risk spectrum, with low and informational taking up more than 90% of the findings. There has been much improvements. The new code reviewed has been written in a clear style that is easier to read and analyze showing continued improvement from the core team.

As of the date of this report, the following comments can be made on the coding style used in the reviewed codebase:

- The codebase is spread over many directories
- The length of source code files is longer than a standard recommendation
- Following the history of the development one can see that there is a code sprawl
- Old code that aren't used any more has not been removed from the repository
- Naming conventions are fluid and not consistent in different parts of the codebase.

The code for development is the same as the one for deployment, which makes the resulting binaries, contain debug functionality, which could lead to exploits.

Even if none of these is related to a single security issue, a common insight in secure development is that clarity breads security. We have found that none of these extraneous items have introduced any security issues, but we do recommend that action be taken to remove abandoned code and reduce debug language in a final production implementation.

1.3.3 Derived Addresses Commentary

There are two uses of derived addresses throughout the code base. First is the address used for tokens and the second is the program address. As we looked through the code it was important to determine which sort of derivation was used depending on the use case. We disclose findings below, but wanted to include commentary here to ensure clarity

In the case where tokens are sent to the wrong address, the only "standard" way to withdraw these tokens out is by signing an instruction with the withdraw authority... but since this authority key has no associated private key, it can't be a signer, therefore the funds are stuck in that account (address). The proposed way under review is to allow another key as authority (signer of the instruction), provided the current value is shown to be derived from that one in a certain way but essentially "retro fitting" a similar allowed way to create an authority which can withdraw. This derivation is using only public info, but since the base key must sign the instruction, someone who does not have the corresponding private key cannot do it. The main risk is that someone else with another key pair can derive the same address, so that he would pretend to be a legitimate withdrawer and steal the money. During our review, even though we disclose findings, we do not believe that it is possible to do a derivation in a manner which would cause a "Collison" meaning the wrong person can withdraw funds.

We believe the derivation has only one seed and we do not see ways of generating collisions with other keys (the problem is essentially a 2nd pre-image problem on SHA256, which cannot be solved with less than 2^128 space or time).

The second derived thing is a "program address", not linked to the above issue, where the code uses multiple seeds, and this is covered by #13 issue below where there is an issue because simple concatenation of seeds is ambiguous, but even if I think it's a conceptual error, the net result is of little value and a low impact.

1.4 Issue Summary List

ID	SEVERITY	FINDING
solana_token_audit#1	High	Missing multisign check
solana_token_audit#2	Informational	Expand the documentation instructions
solana_token_audit#3	Low	Misspelling of type
solana_token_audit#4	Low	Allowing redundant operations
solana_token_audit#5	Informational	Example config file
solana_token_audit#6	Informational	Naming convention of using 2 in method names
solana_token_audit#7	Low	Numerical error casting from f64 to u64
solana_token_audit#8	Low	Numerical error casting from u64 to f64
solana_token_audit#9	Informational	Missing argument support in CLI
solana_token_audit#10	Low	Inconsistent error check order
solana_token_audit#11	Low	Permanently frozen accounts
solana_token_audit#12	Informational	CLI not supporting all instructions
solana_token_audit#13	High	Derived Program Address collisions
solana_token_audit#14	Informational	Derived Address
solana_token_audit#15	Low	Code for checking signing authority

2. METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2 Methodology Flow

2.1 Kickoff

The project is kicked all of the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

2.2 Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

2.3 Review

The review phase is where a majority of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project

3. Assessment of the cryptographic primitives used
4. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

2.4 Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project as a whole. We may conclude that the overall risk is low, but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

2.5 Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

2.6 Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

3. TECHNICAL DETAILS

This section contains the technical details of our findings as well as recommendations for improvement.

3.1 Missing multisign check

Finding ID: solana_token_audit#1

Severity: **High**

Status: **Open**

Description

The validate_owner function does not verify that the signers are unique thus making it allowed for the same signer to sign an instruction multiple time, what this means is that the whole purpose of multi-signing is rendered moot.

Proof of Issue

token/program/src/processor.rs: line 724 (validate_owner)

Filename: token/program/src/processor.rs

Beginning Line Number: 724

Severity and Impact Summary

Example of possible exploit and its impact:

- Account A is going to transfer 1 token to account B. A is configured such that m, the required number of signatures, is 3 and 5 signers (n=5) in total are available (C, D, E, F and G).
- Normally A transfers the token to B by signing with the signatures (C, D, E). What the current implementation permits however is that the transaction instead can be signed 3 times by only using one of these signers, for example (C, C, C).
- Since one signature can be utilized multiple times within a single transaction, the requirements of multi-signing is reduced to having one signature and adding it to the list of signatures m times.

Recommendation

Only count a single signer once within the function, handle repeated cases by either ignoring them or returning an error code.

References

- N/A

3.2 Expand the documentation instructions

Finding ID: solana_token_audit#2

Severity: **Informational**

Status: **Open**

Description

When following along with the example commands, nowhere is it mentioned that one must first use the solana-keygen in order to create an id.json-file before executing spl-token.

Proof of Issue

https://spl.solana.com/token

Severity and Impact Summary

As the documentation is missing, the creation of the keys could be created in a non-secure manner.

Recommendation

Expand the instructions so that the examples are fully working.

References

- https://spl.solana.com/token

3.3 Misspelling of type

Finding ID: solana_token_audit#3

Severity: **Informational**

Status: **Closed**

Description

A spelling error in a type declaration.

Proof of Issue

"type CommmandResult = …" -> "type CommandResult = …"

Filename: token/cli/src/main.rs

Beginning Line Number: 38

Severity and Impact Summary

This is more a problem in that if two or more declarations have the same, or similar names there may be a confusion if the right type is used or not.

Recommendation

Rename the type.

References

- N/A

3.4 Allowing redundant operations

Finding ID: solana_token_audit#4

Severity: **Low**

Status: [**Open**, **Remediated**]

Description

Several operations of dubious value are allowed in the system, should they really be allowed?

- Transfers from and to the same account.
- Minting 0 tokens.
- Burning 0 tokens.
- Assigning the current owner as the new owner of an account.

Proof of Issue

Filename: token/cli/src/main.rs, token/program/src/processor.rs

Beginning Line Number:

Severity and Impact Summary

A Null operation, or an operation that is allowed but has no impact on the state or the effect is void on the blockchain will still have a computational cost. This opens up to a possible DDoS attack where malicious actors would incur cost on the processing of the transactions and thereby could block the processing of legitimate transactions.

Recommendation

If current behavior is not desired, introduce logical checks to verify that the operations are only allowed when they make an actual difference to the accounts.

References

- https://arxiv.org/ftp/arxiv/papers/1912/1912.07497.pdf

3.5 Example config file

Finding ID: solana_token_audit#5

Severity: **Informational**

Status: **Closed**

Description

Since the CLI supports an optional config file, having an example config file somewhere in the repo would be helpful.

Proof of Issue

Lack of config file in repository

Filename: N/A

Beginning Line Number: N/A

Severity and Impact Summary

Nothing, as it more of an informational and pedagogic comment for the future developers.

Recommendation

Include an example config file in the repository.

References

- N/A

3.6 Naming convention of using 2 in method names

Finding ID: solana_token_audit#6

Severity: **Low**

Status: **Closed**

Description

The naming convention of having the instructions be called X and X2 (Approve, Approve2, Transfer, Transfer2 etc.) does not explain anything about the difference between the instructions and makes the system needlessly confusing.

Proof of Issue

Filename: token/program/src/instruction.rs

Beginning Line Number: N/A

Severity and Impact Summary

This is more a problem in that if two or more declarations have unclear, or similar names there may be a confusion if the right method is used or not.

Recommendation

Change the naming convention to better represent the difference between the instructions, e.g. “approve” vs “approve_decimals”

References

- N/A

3.7 Numerical error casting from f64 to u64

Finding ID: solana_token_audit#7

Severity: **Low**

Status: **Closed**

Description

If the ui_amount is a relatively large value, in this case 10^110 and the decimals parameter has the default value of 9, the "expected" result would be 10^20. However, the actual result is u64::MAX.

Proof of Issue

Method ui_amount_to_amount

Filename: token/program/src/lib.rs

Beginning Line Number: 22

Severity and Impact Summary

This error could lead to that the incorrect values are passed to other methods. This could in turn lead to address collisions or the wrong amount being payed between accounts though highly unlikely.

Recommendation

Implement a check confirming that the equation will fit in an u64 or change the type.

References

- https://carols10cents.github.io/rust-conversion-reference/

3.8 Numerical error casting from u64 to f64

Finding ID: solana_token_audit#8

Severity: **Low**

Status: **Closed**

Description

Passing u64::MAX into the function, the result differs slightly since the resulting f64 value cannot completely represent the u64 value creating a rounding error when casting between the types.

Proof of Issue

Method amount_to_ui_amount

Filename: token/program/src/lib.rs

Beginning Line Number: 27

Severity and Impact Summary

This error could lead to that the incorrect values are passed to other methods. This could in turn lead to address collisions or the wrong amount being payed between accounts though highly unlikely.

Recommendation

Take note that some precision can be lost during the conversion and create mitigations for it not impacting the calculations.

References

- https://carols10cents.github.io/rust-conversion-reference/

3.9 Missing argument support in CLI

Finding ID: solana_token_audit#9

Severity: **Informational**

Status: **Open**

Description

CLI is according to the help information supposed to implement --fee-payer and --owner arguments. These features aren't present in the CLI when reviewed.

Proof of Issue

Method main in the referenced files

Filename: token/cli/src/main.rs & token/cli/src/main.rs

Beginning Line Number: 474 & 484

Severity and Impact Summary

The expected functionality is not present.

Recommendation

Implement or remove these features.

References

- N/A

3.10 Inconsistent error check order

Finding ID: solana_token_audit#10

Severity: **Low**

Status: **Closed**

Description

Method process_transfer checks for the MintMismatch case before the AccountFrozen one, most other functions checks for AccountFrozen first.

Proof of Issue

Method process_transfer

Filename: token/program/src/processor.rs

Beginning Line Number: 171

Severity and Impact Summary

The inconsistency in checking for a frozen account could lead to undefined results if something will interfere with the execution. Keeping a consistent structure on the way that the checks are made make the implementation more predictable in its behavior.

Recommendation

Change the order so that AccountFreeze is checked before MintMismatch

References

- N/A

3.11 Permanently frozen accounts

Finding ID: solana_token_audit#11

Severity: **Low**

Status: **Open**

Description

If the freeze_authority is revoked for a token; already frozen accounts becomes permanently frozen.

Proof of Issue

Filename: token/program/src/processor.rs

Beginning Line Number: N/A

Severity and Impact Summary

By forcing a removal of the freeze_authority you can disrupt the validity of the blockchain by first force locking accounts and then remove any possibility to unlock them again. That would lock any account and their balance into the chain without any possibility to access the balance of the locked accounts.

Recommendation

Document whether this is acceptable behavior or not. If it is not, then handle the case when revoking freeze_authority appropriately.

References

- N/A

3.12 CLI not supporting all instructions

Finding ID: solana_token_audit#12

Severity: **Informational**

Status: **Open**

Description

The CLI is missing support for the following instructions:

- Approve
- CloseAccount
- FreezeAccount
- Revoke
- SetAuthority (FreezeAccount).
- SetAuthority (MintToken),
- ThawAccount

Proof of Issue

Filename: token/cli/src/main.rs

Beginning Line Number: N/A

Severity and Impact Summary

The expected functionality is not present.

Recommendation

Either implement these instructions or mention why they are not to be implemented.

References

- N/A

3.13 Derived Program Address collisions

Finding ID: solana_token_audit#13

Severity: **High**

Status: **Open**

Description

A program address is derived by using the SHA256 hash function taking an arbitrary amount of seeds (each seed can have an arbitrary length in the range [0,MAX_SEED_LEN=32)], a public key called program id and the string "ProgramDerivedAddress". All these parameters are fed into the hasher by simple concatenation. The construction of this derivation allows ambiguity in the specification of the seeding material, allowing possible collisions (e.g. same value of the derived program address) with different sets of seeds. For example, the seed sets {"abcdef"}, {"abc", "def"} and {"ab", "cd", "ef"} would all allow derivation of the same program address for the same program ID.

Proof of Issue

solana/sdk/src/pubkey.rs: line 104 (create_program_address)

Filename: solana/sdk/src/pubkey.rs

Beginning Line Number: 104

Severity and Impact Summary

The fact of being able to produce collisions in program addresses by using different strings could be considered a violation of a design principle for derived addresses, i.e. that no easy collisions should be producible with an effort much less than the effort needed to find collisions in the underlying hash function (i.e. this would imply a 128-bit effort for SHA256).

Recommendation

We would suggest moving to a slightly different construction, i.e. hash all the provided seeds independently and then derive the program address by hashing: all the hashes of the seeds, the program id and the string "ProgramDerivedAddress". This simple modification should suffice as fix, while albeit it is not backward compatible with the current implementation.

References

- N/A

3.14 Derivation of an address

Finding ID: solana_token_audit#14

Severity: **Informational**

Status: **Closed**

Description

An address can be derived from a keyed address (base) by using the "create_with_seed" function. This function derives the new address by performing a SHA256 of the base public key, a single seed with length in the range [0, MAX_SEED_LEN=32)] and another public key called owner. All these parameters are fed into the hasher by simple concatenation. Since the size of the two public keys is fixed to 32 bytes, and they are always provided to the function, we do not see a way of producing collisions in the generated address.

An attacker trying to produce the same stake address with his own base public key, is essentially trying to produce a second-preimage for the hash function SHA256. There is currently no method to solve this problem using less than 2^128 effort in terms of either time or storage, see for example the paper [1]. This is coherent with the security level of the chosen signing scheme, i.e. ED25519, which is 128 bits. The fact whether the resulting address is a valid public key for the Ed25519 signature scheme is non influential for the protocol.

Proof of Issue

solana/sdk/src/pubkey.rs: line 88 (create_with_seed)

Filename: solana/sdk/src/pubkey.rs

Beginning Line Number: 88

Severity and Impact Summary

Nothing, informational.

Recommendation

More control on the function could be implemented by avoiding that a potential adversary can play with the size of the seed: an example could be enforce that the length of the seed should always be 32 bytes or at least avoid the zero length, although this is not strictly necessary.

References

- [1] "New Second-Preimage Attacks on Hash Functions" by Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan Hoch, John Kelsey, Adi Shamir & Sébastien Zimmer published in the Journal of Cryptology volume 29, pages657–696(2016).

3.15 Check for signing authority

Finding ID: solana_token_audit#15

Severity: **Low**

Status: **Open**

Description

The authorize function (line 344) is used to change on of the two authorities of a stake account (the staker or the withdrawer key); the function checks that the current authority has the rights to do so. The current withdrawer can change either the withdrawer or the staker, while the staker can only change the staker. The function uses another function called “check” (line 332) to perform the task; however, this happens only for one of the two cases, the other case is managed using the “contains” function.

Proof of Issue

Filename: programs/stake/src/stake_state.rs

Beginning Line Number: 344

Severity and Impact Summary

Increase readability and improve code reuse.

Recommendation

It would be better to uniform the code of the function by exclusively calling the “check” function for both cases.

References

- N/A

APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security

route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

APPENDIX B: DOCUMENT HISTORY

VERSION	STATUS	DATE	AUTHOR	COMMENTS
1.0	Final Report	22 September 2020	Kudelski Team	

APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

SEVERITY	DEFINITION
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.