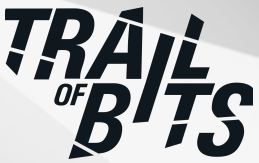

Solang Solana Library

Security Assessment (Summary Report)

July 24, 2023

Prepared for:
Sean Young
Solana Labs

Prepared by: **Vara Prasad Bandaru**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Solana Labs engaged Trail of Bits to review the security of Solang's solana library.

A team of one consultant conducted the review from July 12 to July 19, 2023, for a total of one engineer-weeks of effort. With full access to source code and documentation, we performed manual review of the codebase.

Observations and Impact

As discussed under TOB-SOLLIB-5, there are no tests for the library. Implementing unit tests and integration tests would help the Solang team to identify the issues early on in the development stages.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	2
Low	0
Informational	3
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Configuration	1
Denial of Service	1
Undefined Behavior	3

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Vara Prasad Bandaru, Consultant
vara.bandaru@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
July 12, 2023	Pre-project kickoff call
July 19, 2023	Delivery of report draft
July 24, 2023	Report readout meeting

Project Goals

The engagement was scoped to provide a security assessment of the Solang's Solana library. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do the libraries interact with the correct program?
- Do the libraries encode the instruction data correctly?
- Do the libraries set the required privileges for the accounts?
- Does the spl token library decodes the account data correctly?

Project Targets

The engagement involved a review and testing of the following target.

Solang Solana library

Repository	https://github.com/hyperledger/solang/tree/main/solana-library
Version	`fd366499211c46b1eb2edd5878a371c0b52dd421`
Type	Solidity
Platform	Solana

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Spl token library uses old Token program's Id and supports new instructions	Configuration	**Informational**
2	Insufficient documentation	Undefined Behavior	**Informational**
3	Spl token library incorrectly sets some accounts to writable	Denial of Service	**Medium**
4	Spl token incorrectly decodes the close_authority field of Token account	Undefined Behavior	**Medium**
5	Lack of tests	Undefined Behavior	**Informational**

Detailed Findings

1. Spl token library uses old Token program's Id and supports new instructions	
Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-SOLLIB-1
Target: `solana-library/spl_token.sol`	

Description

The Solang'a `spl_token` library is built to interact with the Token program. There are two versions of the token program, the old program and the new program (2022). The set of instructions supported in the new program is a superset of the instructions supported in the old program.

The library uses the program id of the old program but lists instructions that are only supported in the new program (figure 1.1).

```
address constant tokenProgramId =
address"TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA";
enum TokenInstruction {
	[...]
	InitializeMintCloseAuthority, // 25
	TransferFeeExtension, // 26
	ConfidentialTransferExtension, // 27
	DefaultAccountStateExtension, // 28
	Reallocate, // 29
	MemoTransferExtension, // 30
	CreateNativeMint // 31
}
```

Figure 1.1: solana-library/spl-token.sol#L9–L43

The `tokenProgramId` is of the old token program. However, the listed instructions are only supported in the new program.

The issue does not impact the current version of the library as it does not have functions that use the above instructions. However, adding new functions that use these instructions will impact the contracts using the library; the operations using these functions will fail.

Recommendations

Short term, update the `tokenProgramId` to the new token program Id if the library is intended to use it. Otherwise, remove the instructions that are not supported by the old

Token program. Additionally, document the version of the Token program the library is built to interact with.

Long term, add a mock Token program to the mock `VirtualMachine`, and test the Solana library using the mock Token program. Doing so will help to expose inconsistencies like the ones described here.

2. Insufficient documentation	
Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SOLLIB-2
Target: `solana-library/spl_token.sol`	

Description

The Solang's spl token library contains a fair amount of documentation; however, it lacks areas of documentation necessary for a developer to use the library correctly.

The library would benefit from detailed documentations, especially on:

- The version of the token program the library is built to interact with
- The list of instructions supported by the library
- The supported mode of signing for instructions that allow both single signer owners and multisig owners
- The list of security checks performed by the library before reading data from an account
- The required privileges of the address arguments

Recommendations

Short term, review and properly document the aforementioned aspects of the codebase.

Long term, identify areas of the codebase lacking documentation and add proper developer documentation. Consider creating a usage guide for developers to safely use the library.

3. Spl token library incorrectly sets some accounts to writable

Severity: **Medium**	Difficulty: **Low**
Type: Denial of Service	Finding ID: TOB-SOLLIB-3
Target: `solana-library/spl_token.sol`	

Description

The `spl_token` library sets the `owner` account to `writable` when it is not required to be by the Token program. As a result, the users might provide the `owner` account as a read-only account and the CPI call would fail.

The `Transfer` instruction of the Token program requires three accounts. The third account is the token account owner. The owner account is required to be a signer but not required to be writable.

```
/// Accounts expected by this instruction:
///
///   * Single owner/delegate
///   0. `[writable]` The source account.
///   1. `[writable]` The destination account.
///   2. `[signer]` The source account's owner/delegate.
///   [...]
Transfer {
    /// The amount of tokens to transfer.
    amount: u64,
},
```

Figure 3.1: Declaration of the `Transfer` instruction in the `Token program`

However, the library sets the `owner` account to be `writable` (figure 3.2).

```
function transfer(address from, address to, address owner, uint64 amount) internal {
       instr[0] = uint8(TokenInstruction.Transfer);
      [...]
       AccountMeta[3] metas = [
              [...]
              AccountMeta({pubkey: owner, is_writable: true, is_signer: true})
       ];

       tokenProgramId.call{accounts: metas}(instr);
}
```

Figure 3.2: `transfer` function in `spl_token.sol#L73–L86`

The Solana runtime uses the privileges granted to the caller program to determine what privileges can be extended to the callee. For the contract to supply the `owner` account as `writable` to the Token program, the user calling the contract has to set the account as `writable`.

The user, not aware of this issue, might supply the `owner` account as a read-only account and the call to the Token program would fail because of lack of write privileges. The operations using the function might fail resulting in temporary denial of service.

The following library functions are also vulnerable to this issue:

- `mint_to` sets the authority account to writable
- `burn` function sets the owner account to writable

Exploit Scenario

Bob, a developer, uses the Solang's spl library in his contract. Bob implements a time sensitive operation that uses the `transfer` function of the library. Alice, a user of Bob's contract, calls the time sensitive operation. Alice, being aware of privileges required for the Token program, sets the `owner` account to be a read-only account. The call to the Token program fails because of lack of privileges requested by the library. The execution of the operation fails

Recommendations

Short term, set the `is_writable` flag to `false` for the above mentioned accounts of the instructions.

Long term, thoroughly review the Token program and identify the required privileges for the accounts of the instructions. Ensure that the library provides the accounts with the required privileges without any additional privileges. Implement end-to-end tests for the library.

4. Spl token incorrectly decodes the close_authority field of Token account

Severity: **Medium**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SOLLIB-4
Target: `solana-library/spl_token.sol`	

Description

The library, while computing the `close_authority_present` value, performs comparison with the value `10` instead of `0`. As a result, the `close_authority_present` will be `false` even for the token accounts having a close authority.

The `get_token_account_data` function decodes the data of a token account (figure 4.1).

```
function get_token_account_data(address tokenAccount) public view returns
(TokenAccountData) {
	AccountInfo ai = get_account_info(tokenAccount);

	TokenAccountData data = TokenAccountData(
		{
			mintAccount: ai.data.readAddress(0),
			owner: ai.data.readAddress(32),
			balance: ai.data.readUint64LE(64),
			delegate_present: ai.data.readUint32LE(72) > 0,
			delegate: ai.data.readAddress(76),
			state: AccountState(ai.data[108]),
			is_native_present: ai.data.readUint32LE(109) > 0,
			is_native: ai.data.readUint64LE(113),
			delegated_amount: ai.data.readUint64LE(121),
			close_authority_present: ai.data.readUint32LE(129) > 10,
			close_authority: ai.data.readAddress(133)
		}
	);

	return data;
}
```

Figure 4.1: `get_token_account_data` function in `spl_token.sol#L206-L226`

The `ai.data.readUint32Le(129)` will be 1 if the token account has the `close_authority` and will be `0` otherwise.

Because the function compares against `10`, the `close_authority_present` will be `false` even when the `ai.data.readUint32Le(129)` is 1, i.e. even when the token account has `close_authority`.

This might result in undefined behavior for contracts using the `close_authority` information returned by the library function.

Exploit Scenario

Bob, a developer, uses Solang's `spl_token` library in his protocol. An operation disallows the user to use a Token account having `close_authority`. Bob uses the `get_token_account_data` to read the user supplied token account and determine whether the token account has a `close_authority`.

Eve, an attacker, supplies a Token account having a `close_authority`. The checks implemented by Bob fail to identify that token account has the `close_authority` because of the incorrect data returned by the library. The operation allows Eve's account. Eve takes advantage of this and exploits the protocol.

Recommendations

Short term, change the value used for comparison from `10` to `0`.

Long term, implement comprehensive unit tests for the library. Ensure that the tests cover all execution paths and the code is tested with all kinds of inputs.

5. Lack of tests

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SOLLIB-5
Target: `solana-library/spl_token.sol`	

Description
The codebase does not include any tests for the solana library. Robust unit and integration tests for catching the introduction of certain bugs and logic errors early in the development process. The tests also serve as documentation and an usage guide for the developers.

Exploit Scenario
A bug is found in the solana library. Bob, a developer, uses the library in their protocol. Eve uses the bug in the solana library and exploits Bob's protocol. The bug could have been exposed by thorough unit or integration tests.

Recommendations
Short term, implement comprehensive unit and integration tests for the library. The tests will help expose errors and increase confidence in the functionality of the code.

Long term, regularly compute and review test coverage. Integrate the tests into the CI/CD pipeline.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Update the comment in figure B.1.** The burn operation does not transfer the tokens operation. It burns the specified amount of tokens.

```
/// @param amount the amount to transfer
function burn(address account, address mint, address owner, uint64 amount)
internal {
```

Figure B.1: comment for `amount` parameter of `burn` function in solang/solana-library/spl_token.sol#L93–L94

- **Change the size of `data` value to 3 bytes and set `data[2]` to `0`.** The `SetAuthority` instruction takes an optional `pubkey` value. The Token program ignores the rest of the instruction data if the `pubkey` is absent. The `data[2]` indicates the presence of the public key. Because the `remove_mint_authority` operation does not require a public key, the instruction data could be of 3 bytes with `data[2]` set to `zero`.

```
function remove_mint_authority(address mintAccount, address mintAuthority)
public {
        AccountMeta[2] metas = [...];

        bytes data = new bytes(9);
        data[0] = uint8(TokenInstruction.SetAuthority);
        data[1] = uint8(AuthorityType.MintTokens);
        data[3] = 0;

        tokenProgramId.call{accounts: metas}(data);
}
```

Figure B.2: `remove_mint_authority` function in solang/solana-library/spl_token.sol#L273–L286