HashEye

// SMART CONTRACT SECURITY AUDIT

Solang Solana Library

Security assessment by HashEye · prepared for Solana

HASHEYE AUDITED

PROJECT	Solang Solana Library
CLIENT	Solana
CATEGORY	Solana
PUBLISHED	July 1, 2023
REPORT ID	research-solang-solana-library-2023-07-01-16536d

This report was produced under HashEye's layered review process — **automated detection**, **pattern correlation**, and **senior manual verification** — with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hasheye.io/audits/research-solang-solana-library-2023-07-01-16536d.

Solang Solana Library Security Assessment (Summary Report)

July 24, 2023

Prepared for: Sean Young Solana Labs

Prepared by: Vara Prasad Bandaru

About HashEye Founded in 2012 and headquartered in New York, HashEye 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/hasheye-io/publications, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye 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. HashEye 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 hasheye on Twitter and explore our public repositories at https://github.com/hasheye-io. To engage us directly, visit our "Contact" page at https://www.hasheye.io/contact, or email us at info@hasheye.io. HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 https://www.hasheye.io info@hasheye.io

HashEye 1 Solang Security Assessment

PUBLIC

The sole canonical source for HashEye publications is the HashEye Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic. Test Coverage Disclaimer

All activities undertaken by HashEye 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. HashEye 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.

HashEye 2 Solang Security Assessment

PUBLIC

Table of Contents About HashEye 1

HashEye 3 Solang Security Assessment

PUBLIC

Executive Summary Engagement Overview Solana Labs engaged HashEye to review the security of Solang’s Solana library, which contains the spl_token and system_instruction libraries. One consultant conducted the review from July 12 to July 19, 2023, for a total of one engineer-week of effort. With full access to the source code and documentation, we performed a manual review of the codebase. Observations and Impact During the audit, we uncovered two issues of medium severity in the spl_token library that could either disrupt users’ proper use of the Token Program or give attackers an opportunity to exploit it. First, the spl_token library incorrectly sets the owner account to writable; the Token Program does not require this account to be writable, so users are likely to provide their owner accounts as read-only, which would cause calls to the Token Program to fail (TOB-SOLLIB-3). Second, the library incorrectly decodes the field of the token account that determines whether an account has a close authority; as a result, all token accounts will show that they do not have a close authority, even accounts that do (TOB-SOLLIB-4). This issue could be exploited if the library is used in a context in which close authorities are not allowed. Finally, as discussed in finding TOB-SOLLIB-5, there are no tests for the library. Implementing unit tests and integration tests would help the Solang team to identify issues early on in the development stages.

Recommendations Based on the codebase maturity evaluation and findings identified during the security review, HashEye recommends that the Solang team take the following steps: • Remediate the findings disclosed in this report. These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations. • Create a usage guide for developers. The library should have documentation listing the warnings and notes associated with the library to allow developers to make informed decisions and use the library safely. • Implement unit and integration tests. Ensure the library is thoroughly tested using unit and integration tests.

HashEye 4 Solang Security Assessment

PUBLIC

• Review the Token Program codebase and documentation. Use the Token Program’s code comments and documentation as reference to ensure the library correctly encodes the instruction inputs and decodes data of the token accounts.

Finding Severities and Categories

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

HashEye 5 Solang Security Assessment

PUBLIC

Project Summary Contact Information The following project manager was associated with this project: Jeff Braswell, Project Manager jeff.braswell@hasheye.io The following engineering director was associated with this project: Josselin Feist, Engineering Director, Blockchain josselin.feist@hasheye.io The following engineer was associated with this project: Vara Prasad Bandaru, Consultant vara.bandaru@hasheye.io 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 July 24, 2023 Delivery of comprehensive report

HashEye 6 Solang Security Assessment

PUBLIC

Project Goals The engagement was scoped to provide a security assessment of 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 correctly encode the instruction data? • Do the libraries correctly set the required privileges for the accounts? • Does the spl_token library decode the account data correctly?

HashEye 7 Solang Security Assessment

PUBLIC

Project Targets The engagement involved a review and testing of the following target. Solang Solana Library Repository https://github.com/hyperledger/solang Version fd366499211c46b1eb2edd5878a371c0b52dd421 Subdirectory solana-library Type Solidity Platform Solana

HashEye 8 Solang Security Assessment

PUBLIC

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 but 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 the token account Undefined Behavior Medium 5 Lack of tests Undefined Behavior Informational

HashEye 9 Solang Security Assessment

PUBLIC

Detailed Findings 1. spl_token library uses old Token Program’s ID but supports new instructions Severity: Informational Difficulty: Low Type: Configuration Finding ID: TOB-SOLLIB-1 Target: solana-library/spl_token.sol

Description Solang’s 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 supported only 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 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 would impact contracts using the library; the operations using such functions would 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 that 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 one described here .

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

Description Solang’s spl_token library contains a fair amount of documentation; however, it lacks areas of documentation necessary for developers to correctly use the library. The library would benefit from detailed documentation, especially on the following: • The version of the Token Program that the library is built to interact with • The instructions supported by the library • The supported mode of signing for instructions that allow both single-signer owners and multisig owners • The 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 that lack documentation and add proper developer documentation. Consider creating a usage guide for developers to safely use the library.

3. spl_token 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 even though it is not required to be by the Token Program. As a result, if users provide the owner account as a read-only account, CPI calls will 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 is 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: The declaration of the Transfer instruction in the Token Program However, the library sets the owner account to 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: The 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. A user who is 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 the lack of write privileges. The operations using the function might fail, resulting in a temporary denial of service. The following library functions are also vulnerable to this issue: • mint_to, which sets the authority account to writable • burn, which sets the owner account to writable Exploit Scenario Bob, a developer, uses the Solang spl_token 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 is aware of the privileges required for the Token Program, so she sets the owner account to be a read-only account. The call to the Token Program fails because of the 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 owner/authority account of the above-mentioned instructions/functions. 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 more end-to-end tests for the library.

4. spl_token incorrectly decodes the close_authority field of the token account Severity: Medium Difficulty: Medium Type: Undefined Behavior Finding ID: TOB-SOLLIB-4 Target: solana-library/spl_token.sol

Description The spl_token library, while computing the close_authority_present value, performs a comparison with the value 10 instead of 0. As a result, the close_authority_present value will be false even for token accounts that have 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: The get_token_account_data function in spl_token.sol#L206-L226 The ai.data.readUint32Le(129) value will be 1 if the token account has the close_authority field; otherwise, it will be 0. Because the function compares ai.data.readUint32Le(129) against 10, the close_authority_present value will be false even when the

ai.data.readUint32Le(129) is 1 (i.e., even when the token account has a 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 users from using token accounts that have a close authority. Bob uses the get_token_account_data function to read the user-supplied token account and determine whether it has a close authority. Eve, an attacker, supplies a token account that has a close authority. The checks implemented by Bob fail to identify that the token account has a 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 the comparison from 10 to 0. Long term, implement comprehensive unit tests for the library. Ensure that the tests cover all execution paths and that 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 includes limited tests for the Solana library. Robust unit and integration tests are important for catching the introduction of certain bugs and logic errors early in the development process. They also serve as documentation and usage guides for developers. Exploit Scenario A bug is found in the Solana library. Bob, a developer, uses the library in his 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. Such tests will help expose errors and increase confidence in the functionality of the code. Long term, regularly compute and review the 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, implementing them may 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 specified number of tokens; it burns them.

/// @param amount the amount to transfer function burn(address account, address mint, address owner, uint64 amount) internal { Figure B.1: The comment for the amount parameter of the burn function in solang/solana-library/spl_token.sol#L93-L94 • Change the size of the SetAuthority instruction's 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

value is absent. The use of data[2] indicates the presence of a public key. Because the remove_mint_authority operation does not require a public key, the instruction's data value could be 3 bytes with data[2] set to 0.

```
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: The remove_mint_authority function in solang/solana-library/spl_token.sol#L273-L286