// Solana Program Security Assessment 09.09.2024 - 09.25.2024

Solana Hypergrid

Sonic

/ DRAFT /

HALBORN

Solana Hypergrid - Sonic

Prepared by: HALBORN

Last Updated 09/25/2024

Date of Engagement by: September 9th, 2024 - September 25th, 2024

Summary

25% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	0	4

TABLE OF CONTENTS

1. Introduction

Sonic engaged Halborn to conduct a security assessment on their Hypergrid program beginning on September 9th, 2024 and ending on September 25th, 2024. The security assessment was scoped to the smart contracts provided in the GitHub repository hypergrid-grid. Commit hashes, and further details, can be found in the Scope section of this report.

Sonic is releasing a new version of Hypergrid, which includes functionality to synchronize accounts between the Solana Base Layer network and the Hypergrid Shared State Network (HSSN).

2. Assessment Summary

Halborn was provided 2.5 weeks for the engagement and assigned one full-time security engineer to review the security of the Solana Program in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the codebase.
- Check that the codebase does not have any known vulnerability that might affect the participants funds
- Validate that codebase properly implements the functionalities to keep the base layer synchronized with the HSSN, such as identifying remote accounts properly.

In summary, Halborn did not identify any significant security risks, but some informational findings should be addressed by the Sonic team. The main ones were:

- Fee multiplier env parameter value is not validated
- Hardcoded keypair in config file
- Lack of clarity for node roles values
- Meaningless regex value to capture events

3. Test Approach And Methodology

`Halborn` performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the token airdrop program.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (`cargo audit`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: hypergrid-grid

(b) Assessed Commit ID: 28de26f

(c) Items in scope:

- account-decoder/src/lib.rs
- accounts-db/src/accounts_cache.rs
- accounts-db/src/accounts_db.rs
- accounts-db/src/inline_spl_token_2022.rs
- accounts-db/src/read_only_accounts_cache.rs
- accounts-db/src/remote_loader.rs
- hypergrid/src/cosmos.rs
- hypergrid/src/lib.rs
- hypergrid/src/remote_loader.rs
- program-runtime/src/message_processor.rs
- programs/sonic-account-migrater/src/lib.rs
- programs/sonic-account-migrater/src/processor.rs
- programs/sonic-fee-settlement/src/lib.rs
- programs/sonic-fee-settlement/src/processor.rs
- rpc-client-api/src/config.rs
- rpc/src/rpc.rs
- rpc/src/rpc_pubsub.rs
- rpc/src/rpc/account_resolver.rs
- runtime/src/bank.rs
- runtime/src/builtins.rs
- sdk/program/src/lib.rs
- sdk/program/src/sonic_account_migrater/instruction.rs
- sdk/program/src/sonic_account_migrater/mod.rs
- sdk/program/src/sonic_fee_settlement/instruction.rs
- sdk/program/src/sonic_fee_settlement/mod.rs
- sdk/program/src/sonic_fee_settlement/state.rs
- sdk/src/account.rs
- sdk/src/fee.rs
- sdk/src/lib.rs
- hypergrid/src/config.rs

Out-of-Scope:

REMEDIATION COMMIT ID:

- https://github.com/mirrorworld-universe/hypergrid-grid/pull/65/files

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - FEE MULTIPLIER ENV PARAMETER VALUE IS NOT VALIDATED	INFORMATIONAL	-
HAL-02 - HARDCODED KEYPAIR IN CONFIG FILE	INFORMATIONAL	SOLVED - 09/15/2024
HAL-03 - LACK OF CLARITY FOR NODE ROLES VALUES	INFORMATIONAL	-
HAL-04 - MEANINGLESS REGEX VALUE TO CAPTURE EVENTS	INFORMATIONAL	-

7. FINDINGS & TECH DETAILS

7.1 (HAL-01) FEE MULTIPLIER ENV PARAMETER VALUE IS NOT VALIDATED

// INFORMATIONAL

Description

The `fee multiplier` is a parameter to obtain the `congestion multiplier`, which calculation is shown in the snippet below:

sdk/src/fee.rs

```
98          // Fee based on compute units and signatures
99          let congestion_multiplier = if lamports_per_signature == 0 {
100             0.0 // test only
101         } else {
102             // 1.0 // multiplier that has no effect
103             //Sonic: custom congestion multiplier
104             self.fee_multiplier as f64 / 10000 as f64
105         };
```

The `congestion multiplier` is a number used to increase the transaction fee a number of times in case that exists a high congestion in the network, as shown in the snippet below:

sdk/src/fee.rs

```
138         ((budget_limits
139             .prioritization_fee
140             .saturating_add(signature_fee)
141             .saturating_add(write_lock_fee)
142             .saturating_add(compute_fee) as f64)
143             * congestion_multiplier)
144             .round() as u64
```

The fee multiplier can be set through an environment parameter called `SONIC_FEE_MULTIPLIER` . In case that the mentioned environment parameter is not found, the default value is then `10000` . This behavior can be seen in the snippet below:

sdk/src/fee.rs

```
53          let fee_multiplier: Option<&'static str> = option_env!("SONIC_F
54          let fee_multiplier = match fee_multiplier.unwrap_or("10000").pa
```

```
55              Ok(f) => f,
56              Err(_) => 10000,
57          };
58
59          FeeStructure {
60              lamports_per_signature: sol_to_lamports(sol_per_signature),
61              lamports_per_write_lock: sol_to_lamports(sol_per_write_lock
62              compute_fee_bins,
63              fee_multiplier
64          }
```

The snippet above highlights the value expected for the `fee multiplier` .
The `fee multiplier` is expected to fall between 2 boundaries, a max and a min.
The current implementation of the `hypergrid-grid` program does not validate that the parameter provided through environment parameter complies with a max and min value.
As a result, depending on the value of the `fee multiplier`, the behavior can be:

- if the `fee multiplier` is too high, the transaction costs can be too high to be paid
- if the `fee multiplier` is too low, the transaction costs can result in a value close to 0

Score

Impact:

Likelihood:

Recommendation

Consider adding a validation on the env parameter reading, which checks that the provided value does not overpass a max value and does not surpass a min value.

7.2 (HAL-02) HARDCODED KEYPAIR IN CONFIG FILE

// INFORMATIONAL

Description

The `config.rs` file currently has a hardcoded keypair, as shown in the snippet below:

hypergrid/src/config.rs

```
28  impl Default for Config {
29      fn default() -> Self {
30          let keypair_base58 = "5gA6JTpFziXu7py2j63arRUq1H29p6pcPMB74LaNu
31          let baselayer_rpc_url = "https://api.devnet.solana.com".to_stri
32          let sonic_program_id ="4WTUyXNcf6QCEj76b3aRDLPewkPGkXFZkkyf3A3v
33          let hssn_rpc_url: String = "http://localhost:1317".to_string();
34
35          Self {
36              baselayer_rpc_url,
37              hssn_rpc_url,
38              keypair_base58,
39              sonic_program_id,
40          }
41      }
42  }
```

After investigating about the corresponding public key in Solana block explorers, there couldn't be found any evidence that it was used in production i.e. used to deploy programs or manage funds in mainnet. Although we couldn't find any associated risk to the use of the mentioned hardcoded keypair, it is highly recommended not to store any kind of private key in plain text, even more considering that the keypair is publicly accessible through the `hypergrid-grid` repository.

Score

Impact:

Likelihood:

Recommendation

Delete any reference to the mentioned key pair from the codebase.
It is also highly recommended to deprecate the key pair of being used in any kind of operation, even if it is not used for production purposes, since it is already publicly available in the `hypergrid-grid` repository.

Remediation

SOLVED: The **Sonic team** solved this issue by replacing the hardcoded keypair with a snippet to access the private key through a keypair file.

Remediation Hash

https://github.com/mirrorworld-universe/hypergrid-grid/pull/65/files

7.3 (HAL-03) LACK OF CLARITY FOR NODE ROLES VALUES

// INFORMATIONAL

Description

The `HypergridNode` structure describes some key parameters of a Hypergrid Node, as shown in the snippet below:

hypergrid/src/remote_loader.rs

```
24  #[derive(Debug, Default)]
25  struct HypergridNode {
26      pub pubkey: Pubkey,
27      pub name: String,
28      pub rpc: String,
29      pub role: i32,
30  }
```

One of the key parameters that we can observe si the `role` field. Currently, the `role` parameter is being used by the `load_account_via_rpc` function to decide the value fo the `rpc_url` , as shown in the snippet below:

hypergrid/src/remote_loader.rs

```
205         let mut rpc_url = self.config.baselayer_rpc_url.clone();
206         if let Some(source) = source {
207             if self.hypergrid_nodes.len() < 1 || self.hypergrid_nodes.g
208                 self.load_hypergrid_nodes();
209             }
210             if let Some(node) = self.hypergrid_nodes.get(&source) {
211                 if node.value().role == 2 || node.value().role == 3 ||
212                     rpc_url = node.value().rpc.clone();
213                 } else {
214                     info!("load_account_via_rpc: invalid source role: {
215                     return None;
216                 }
217             }
218         }
```

After carefully investigating throughout the codebase to have an understanding of the meaning of these roles, there couldn't be find any comment on it.

In order to improve code readability, it is recommended to add as many comments as possible, speciallly in cases where the role of an actor of the system is interpreted on code to take actions.

Score

Impact:

Likelihood:

Recommendation

Consider adding comments to the code explaining the different roles and the meaning of every of them. Alternatively, and to improve code readability, consider implementing the use of an enum to clearly state the name and meaning of the different roles.

7.4 (HAL-04) MEANINGLESS REGEX VALUE TO CAPTURE EVENTS

// INFORMATIONAL

Description

The `post_status_to_baselayer` function is used to communicate an state change to the target base layer, in this case, Solana.
In order to do so, the function takes the log messages of a transaction and looks for an specific event through the use of a regular expression, as shown in the snippet below:

runtime/src/bank.rs

```
5006            let mut signature: Option<Signature> = None;
5007            log_messages.as_ref().map(|log_messages| {
5008                let re = Regex::new(r"Fake NFT New Value: (\d+)").unwrap();
5009                for log_message in log_messages.iter() {
5010                    info!("log_message: {:?}", log_message);
5011
5012                    //Sonic: send states to baselayer
5013                    let caps = re.captures(log_message);
5014                    if let Some(caps) = caps {
5015                        let value = caps.get(1).map_or("", |m| m.as_str());
5016
5017                        signature = remote_loader.send_status_to_baselayer(
5018                        break;
5019                    }
5020                }
5021            });
```

As the snippet above highlights, the regular expression used to look for events to post a new status to the base layer is `Fake NFT New Value` which lacks clarity on the real purpose of the event emitted.
In order to improve code readability, and to improve the information content in the expected event, is that is necessary to update both the event emitted to post status to the base layer, and to update the regex to catch the event acccordingly.

Score

Impact:

Likelihood:

Recommendation

Consider changing both the event emitted and the regular expression to a value that is meaningful and self-explanatory.

8. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	PACKAGE	SHORT DESCRIPTION
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on èd25519-dalek`

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.