// HALBORN

Solana Foundation - Durable Nonce Patch

L1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 6th, 2022 - June 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/10/2022	Piotr Cielas
0.2	Draft Review	06/10/2022	Gabi Urrutia
1.0	Remediation Plan	06/15/2022	Piotr Cielas
1.1	Remediation Plan Review	06/15/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Solana Foundation engaged Halborn to conduct a security audit on their pull requests, patching the Durable Nonce runtime bug beginning on June 6th, 2022 and ending on June 9th, 2022 .

Sealevel, Solana's parallel smart contracts runtime, can process transactions in parallel because Solana transactions describe all the states a transaction will read or write while executing. This not only allows for non-overlapping transactions to execute concurrently, but also for transactions that are only reading the same state to execute concurrently as well.

This security assessment was scoped to the implementation of the runtime available in the solana GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

On-chain components were prioritized in this audit.

1.2 AUDIT SUMMARY

The team at Halborn was provided 3 days for the engagement and assigned 1 full-time security engineer to audit the security of the code in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues within the pathes in scope.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which should be addressed. The main ones are the following:
- It is recommended not to use the unwrap function in the production

environment because its use causes panic! and may crash the contract without verbose error messages.
- Consider introducing new types with verbose names to better explain the meaning of variables values.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, 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 audit:

- Research into the architecture, purpose, and use of the platform.
- Manual program code review and walkthrough to identify 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.
- Local cluster deploying (solana-test-validator)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that

were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Code repositories:

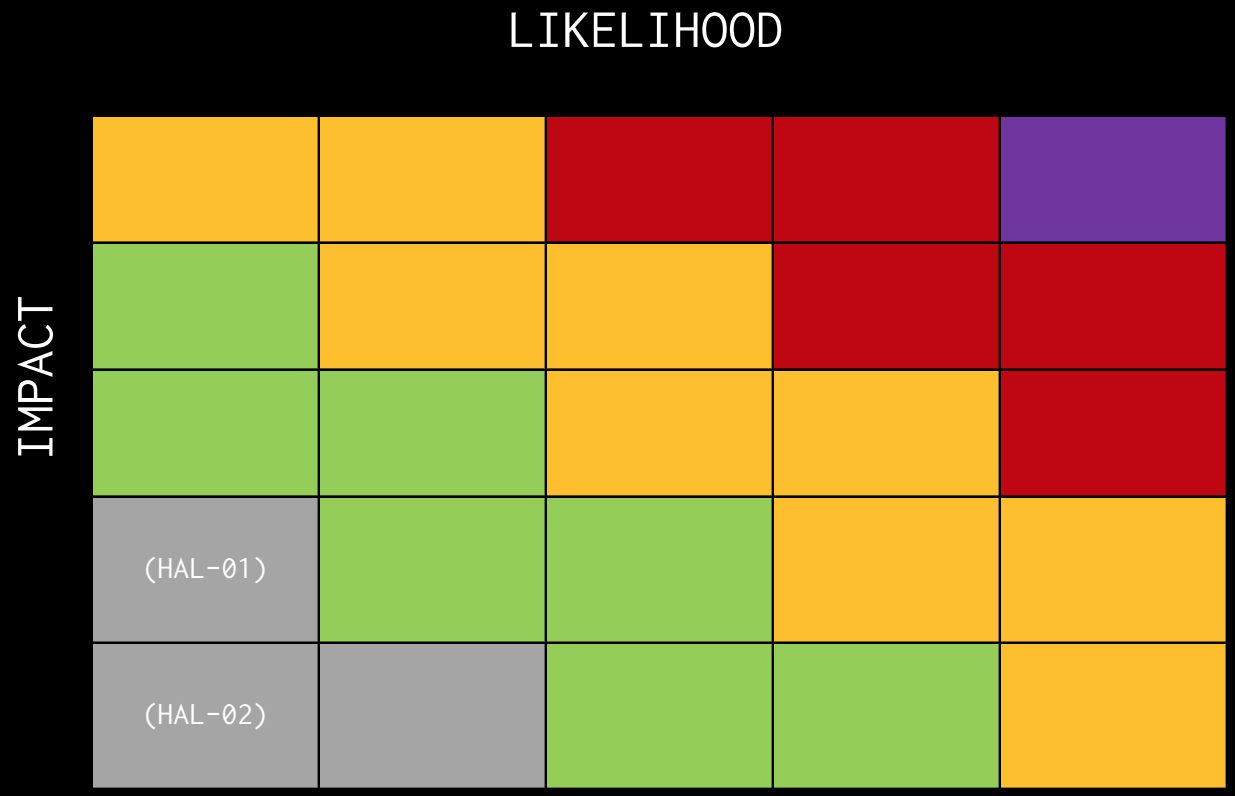
1. Sealevel runtime

- Repository: solana
- Pull requests in scope:
    1. #25744
    2. #25788
    3. #25789
    4. #24396
    5. #25831

Out-of-scope: External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SUSCEPTIBLE TO RUST PANICS DUE TO UNSAFE UNWRAP USAGE	Informational	ACKNOWLEDGED
CONFUSING FUNCTION CALL CONVENTION	Informational	FUTURE RELEASE

FINDINGS & TECH DETAILS

3.1 (HAL-01) SUSCEPTIBLE TO RUST PANICS DUE TO UNSAFE UNWRAP USAGE - INFORMATIONAL

Description:

Pull Request: #25788

The use of helper methods in Rust, such as unwrap, is allowed in dev and testing environment because those methods are supposed to throw an error (also known as panic!) when called on Option::None or a Result which is not Ok. However, keeping unwrap functions in production environment is considered bad practice because they may lead to program crashes, which are usually accompanied by insufficient or misleading error messages.

Code Location:

Listing 1: runtime/src/accounts.rs (Lines 1337,1345)

```
1337 let nonce_versions = StateMut::<NonceVersions>::state(nonce.
  ↳ account()).unwrap();
1338 if let NonceState::Initialized(ref data) = nonce_versions.state()
  ↳ {
1339     let nonce_state = NonceState::new_initialized(
1340         &data.authority,
1341         durable_nonce,
1342         lamports_per_signature,
1343     );
1344     let nonce_versions = NonceVersions::new(nonce_state,
  ↳ separate_domains);
1345     account.set_state(&nonce_versions).unwrap();
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended not to use the unwrap function in the production environment because its use causes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability and, in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrapping, or using the error-chain crate for errors.

Remediation Plan:

ACKNOWLEDGED: Pull requests did not introduce or modify those unwraps in scope of this audit. Additionally, comments in the code state “Since we know we are dealing with a valid nonce account unwrap is safe here”.

3.2 (HAL-02) CONFUSING FUNCTION CALL CONVENTION - INFORMATIONAL

Description:

Pull Request: #25788

The patch changes definitions of several functions which previously required the caller to provide a DurableNonce and now require a tuple with a DurableNonce and a boolean, indicating the separation of blockhash and nonce domains.

If this boolean is set to true, DurableNonces are generated from a hash of the most recent blockhash and a fixed seed to prevent transaction replay.

This boolean parameter used is in multiple function calls across the codebase and is usually accompanied by a comment. This inconsistency might be confusing to developers unfamiliar with the bug and previous nonce format, which may lead to them writing incorrect code.

Code Location:

Listing 2: runtime/src/accounts.rs (Line 1199)

```
1196 res: &'a [TransactionExecutionResult],
1197 loaded: &'a mut [TransactionLoadResult],
1198 rent_collector: &RentCollector,
1199 durable_nonce: &(DurableNonce, /*separate_domains:*/ bool),
1200 lamports_per_signature: u64,
1201 leave_nonce_on_success: bool,
```

Listing 3: runtime/src/accounts.rs (Line 1231)

```
1228 execution_results: &'a [TransactionExecutionResult],
1229 load_results: &'a mut [TransactionLoadResult],
1230 rent_collector: &RentCollector,
1231 durable_nonce: &(DurableNonce, /*separate_domains:*/ bool),
```

```
1232 lamports_per_signature: u64,
1233 leave_nonce_on_success: bool,
```

Listing 4: runtime/src/accounts.rs (Line 1318)

```
1315     execution_result: &Result<()>,
1316     is_fee_payer: bool,
1317     maybe_nonce: Option<(&'a NonceFull, bool)>,
1318     &(durable_nonce, separate_domains): &(DurableNonce, bool),
1319     lamports_per_signature: u64,
1320 ) -> bool {
```

Listing 5: runtime/src/nonce_keyed_account.rs (Line 19)

```
19 fn get_durable_nonce(invoke_context: &InvokeContext) -> (
↳ DurableNonce, /*separate_domains:*/ bool) {
20     let separate_nonce_from_blockhash = invoke_context
21         .feature_set
22         .is_active(&feature_set::separate_nonce_from_blockhash::id
↳ ());
23     let durable_nonce =
24         DurableNonce::from_blockhash(&invoke_context.blockhash,
↳ separate_nonce_from_blockhash);
25     (durable_nonce, separate_nonce_from_blockhash)
```

Listing 6: runtime/src/nonce_keyed_account.rs (Line 19)

```
19 fn get_durable_nonce(invoke_context: &InvokeContext) -> (
↳ DurableNonce, /*separate_domains:*/ bool) {
20     let separate_nonce_from_blockhash = invoke_context
21         .feature_set
22         .is_active(&feature_set::separate_nonce_from_blockhash::id
↳ ());
23     let durable_nonce =
24         DurableNonce::from_blockhash(&invoke_context.blockhash,
↳ separate_nonce_from_blockhash);
25     (durable_nonce, separate_nonce_from_blockhash)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider introducing a new type with a verbose name to better explain the meaning of the `separate_domains` variable value.

Remediation Plan:

PENDING: This is being tracked in a GitHub issue.

THANK YOU FOR CHOOSING

// HALBORN