Vector Bridge
Avail

HALBORN

Vector Bridge - Avail

Prepared by: HALBORN

Last Updated 07/17/2024

Date of Engagement by: April 17th, 2024 - May 3rd, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
7	0	0	1	1	5

TABLE OF CONTENTS

8. Automated Testing

1. Introduction

Avail engaged Halborn to conduct a security assessment on their smart contracts beginning on April 17th, 2024 and ending on May 3rd, 2024. The security assessment was scoped to crates provided in the GitHub repositories avail and avail-core, commit hashes, and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at Halborn was provided 2 weeks for the engagement and assigned two full-time security engineers to check the security of crates. The security engineers are blockchain and smart-contract security experts with advanced penetration testing and smart-contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the avail codebase

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks that were addressed and accepted by the `Avail team`. The main ones were the following:

- `Incorporate maximum size validation checks within the MemoryTemporaryStorage struct.`
- `Integrate thorough checks to prevent insufficient block dimensions and mitigate any risk of data loss.`

3. Test Approach And Methodology

`Halborn` performed a combination of the manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance the coverage of smart contracts. They 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 architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Race condition tests.
- Cross contract call controls.
- Architecture related logical controls.
- Fuzz testing (`cargo-fuzz`).
- Test coverage review (`cargo tarpaulin`).
- Scanning of Rust files for vulnerabilities (`cargo audit`).
- Checking the unsafe code usage (`cargo-geiger`).

4. 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: avail

(b) Assessed Commit ID: af54545

(c) Items in scope:

- availproject/avail/pull/427
- availproject/avail-core/pull/73/

Out-of-Scope: Third party dependencies., Economic attacks.

FILES AND REPOSITORY

(a) Repository: avail-core

(b) Assessed Commit ID: eb5e631

(c) Items in scope:

- avail-core/core
- avail-core/kate
- avail-core/kate/recovery
- availproject/avail/pull/427
- availproject/avail-core/pull/73/

Out-of-Scope: Third party dependencies., Economic attacks.

REMEDIATION COMMIT ID:

- https:/https:/
- https:/https:/

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	5

IMPACT X LIKELIHOOD

	HAL-01			
			HAL-02	
HAL-03 HAL-04 HAL-05 HAL-06 HAL-07				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING VALIDATION COULD LEAD TO INADEQUATE BLOCK DIMENSIONS AND POTENTIAL DATA LOSS	MEDIUM	SOLVED - 07/11/2024
MISSING MAXIMUM SIZE VALIDATION	LOW	RISK ACCEPTED
PRESENCE OF TODOS AND COMMENTS IMPLYING UNFINISHED CODE	INFORMATIONAL	PARTIALLY SOLVED - 06/27/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
STYLISTIC IMPROVEMENTS	INFORMATIONAL	SOLVED - 06/27/2024
GAS OPTIMIZATIONS	INFORMATIONAL	ACKNOWLEDGED
ERRONEOUS COMMENTS	INFORMATIONAL	PARTIALLY SOLVED - 06/27/2024
TYPOS IN COMMENTS	INFORMATIONAL	PARTIALLY SOLVED - 06/27/2024

7. FINDINGS & TECH DETAILS

7.1 MISSING VALIDATION COULD LEAD TO INADEQUATE BLOCK DIMENSIONS AND POTENTIAL DATA LOSS

// MEDIUM

Description

The `get_block_dimensions` function efficiently calculates the optimal block dimensions for a matrix-style block, considering both the `block_size` and `chunk_size`, which define the data chunk per matrix cell.
It is mentioned in the comments that "*Both row number and column number have to be a power of 2, because of the Plonk FFT constraints*" and that "*the total_cells number will also be a power of 2*" but neither of those assumptions were explicitly validated.
Given that `total_cells` is determined as the nearest power of 2 to the `block_size` divided by `chunk_size`, a scenario arises: if `total_cells` surpasses the maximum column count and isn't a power of 2 (which occurs when `chunk_size` isn't a power of 2), then the remainder of the division between `total_cells` and the maximum column count won't be zero, expressed as `total_cells % nz_max_cols != 0`.
As a result, the block dimensions will be one row short, as the row count equals `total_cells / nz_max_cols`. Consequently, some transaction data might be omitted from the block matrix.

Code Location

The `get_block_dimensions` function is located in the `kate/src/com.rs` file and is defined as follows:

```
207 pub fn get_block_dimensions(
208     block_size: u32,
209     max_rows: BlockLengthRows,
210     max_cols: BlockLengthColumns,
211     chunk_size: NonZeroU32,
212 ) -> Result<BlockDimensions, Error> {
213     let max_block_dimensions =
214         BlockDimensions::new(max_rows, max_cols, chunk_size).ok_or(Error:
215     let max_block_dimensions_size = max_block_dimensions.size();
216
217     let block_size = usize::try_from(block_size)?;
218     ensure!(block_size <= max_block_dimensions_size, Error::BlockTooBig);
219
220     if block_size == max_block_dimensions_size || MAXIMUM_BLOCK_SIZE {
221         return Ok(max_block_dimensions);
222     }
223
224     // Both row number and column number have to be a power of 2, because
225
```

```
226     // Implicitly, if both of the assumptions above are correct, the tota
227     let mut nearest_power_2_size = 2_usize.pow((block_size as f32).log2()
228     if nearest_power_2_size < MINIMUM_BLOCK_SIZE {
229         nearest_power_2_size = MINIMUM_BLOCK_SIZE;
230     }
231
232     let total_cells = (nearest_power_2_size as f32 / chunk_size.get() as
233
234     // we must minimize number of rows, to minimize header size
235     // (performance wise it doesn't matter)
236     let nz_max_cols = NonZeroU32::new(max_cols.0).ok_or(Error::ZeroDimens
237     #[allow(clippy::arithmetic_side_effects)]
238     let (cols, rows) = if total_cells > max_cols.0 {
239         (max_cols, BlockLengthRows(total_cells / nz_max_cols))
240     } else {
241         (BlockLengthColumns(total_cells), BlockLengthRows(1))
242     };
243
244     BlockDimensions::new(rows, cols, chunk_size).ok_or(Error::BlockTooBig
}
```

Proof of Concept

The following PoC exploit illustrates two scenarios where data chunks were lost due to miscalculations in block dimensions. Here is an elaborate explanation of the first scenario, considering the following parameters:

- `block_size` = 128 (2^7)
- `max_rows` = 2 (2^1)
- `max_cols` = 4 (2^2)
- `chunk_size` = 31

In this case, `nearest_power_2_size` also equals 128, and `total_cells` equals the ceiling of `128.0 / 31.0 = 4.129`, which rounds up to `5`.

Given that `total_cells > max_cols` (since 5 > 4), the block's dimensions will be `max_cols, BlockLengthRows(total_cells / nz_max_cols)`, effectively `4, 5/4`. This means the block's dimensions will accommodate `4*1` cells instead of 5, leading to the loss of `128 - 4*31 = 4` data chunks.

Consider integrating the following PoC exploit code into the `kate/src/com.rs` file alongside the existing tests.

```
#[cfg(not(feature = "maximum-block-size"))]
#[test_case(128, 2, 4, 31 => (1, 4, 31) ; "total cells equals 124 which is les
#[test_case(8000, 256, 256, 31 => (1, 256, 31) ; "total cells equals 7936 whic
fn test_poc_get_block_dimensions(
    size: u32,
    rows: u32,
```

```
    cols: u32,
    chunk_size: u32,
) -> (u32, u32, u32) {
    let dims = get_block_dimensions(
        size,
        BlockLengthRows(rows),
        BlockLengthColumns(cols),
        unsafe { NonZeroU32::new_unchecked(chunk_size) },
    )
    .unwrap();

    (dims.rows.0, dims.cols.0, dims.chunk_size.get())
}
```

Executing the aforementioned PoC produces the subsequent outcome:

```
successes:
    com::tests::test_poc_get_block_dimensions::total_cells_equals_124_which_is_less_than_128
    com::tests::test_poc_get_block_dimensions::total_cells_equals_7936_which_is_less_than_8000

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 80 filtered out; finished in 0.00s
```

Score

Impact: 3

Likelihood: 4

Recommendation

Consider one of the following options:

- Increment the row's count by one if `total_cells > max_cols.0` and `total_cells % nz_max_cols != 0`.
- Validate that `chunk_size`, `max_rows`, and `max_cols` are powers of 2 if that's the expected behavior.

Remediation Plan

SOLVED: The **Avail team** addressed this issue by defining `chunk_size` as a constant set to 32 (2^5) and adding the required checks to ensure that both `max_rows` and `max_cols` are powers of 2.

Remediation Hash

https://github.com/availproject/avail-core/pull/100/commits/cbbf09a69d76a46bb685a85169f0076e884c90cb

References

availproject/avail-core/kate/src/com.rs#L238

7.2 MISSING MAXIMUM SIZE VALIDATION

// LOW

Description

`MemoryTemporaryStorage` handles key-value storage in memory, including operations to get, insert, remove, take, update, and clear storage items which are later on used by `post_inherent_data` operations
There are 2 Main concerns about the implementation of the `MemoryTemporaryStorage`
1. The current `insert` method does not incorporate checks to validate the size of encoded values before insertion. This omission can lead to Out-of-Memory (OOM) conditions if large data objects are processed.
2. There is no mechanism to control or limit the number of elements that can be inserted into the storage. Without such constraints, there is a potential for unbounded growth in memory usage, which can severely degrade performance and lead to memory exhaustion.
These deficiencies are particularly concerning when it comes to the usage of memory storage in the context of blockchain systems to avoid possible OOM attacks in the future.

BVSS

AO:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:N/S:C (4.1)

Recommendation

- Introduce validation checks that assess the size of data prior to its insertion into the storage system. Establishing a maximum permissible size for each entry will help mitigate the risk of OOM conditions.
- Define and enforce a cap on the number of elements permissible within the storage.

Remediation Plan

RISK ACCEPTED: The **Avail team** accepted the risk of this issue, based on the internal tests they conducted, `MemoryTemporaryStorage` is only available for pallet creators.

References

availproject/avail/pull/427
availproject/avail/pull/427

7.3 PRESENCE OF TODOS AND COMMENTS IMPLYING UNFINISHED CODE

// INFORMATIONAL

Description

Open TODOs can point to architecture or programming issues that still need to be resolved.

Code Location

The `core/src/constants.rs` file contains the following TODO comment:

```
13 | // TODO: evaluate whether we should consider moving this into avail
```

The `kate/src/com.rs` file contains the following TODO comments:

```
288 | // TODO: Better error type for BlsScalar case?
```

```
360 | //TODO cache extended data matrix
361 | //TODO explore faster Variable Base Multi Scalar Multiplication
```

```
435 | // row has to be a power of 2, otherwise interpolate() function panics TO
```

The `kate/recovery/src/commitments.rs` file contains the following TODO comment:

```
116 | // @TODO Opening Key here???
```

The `kate/recovery/src/com.rs` file contains the following comment:

```
580 | // @note the way it's done should be improved
```

The `kate/recovery/src/config.rs` file contains the following TODO comment:

```
  1 | // TODO: Constants are copy from kate crate, we should move them to commo
```

The `kate/src/gridgen/mod.rs` file contains the following TODO comment:

```
277 | // TODO: fix this all up without the gross conversions after moving to ar
```

Score

(0.0)

Recommendation

Consider resolving the TODOs before deploying.

Remediation Plan

PARTIALLY SOLVED: The **Avail team** addressed most of the mentioned TODOs and comments, leaving only the last TODO comment unresolved.

Remediation Hash

https://github.com/availproject/avail-core/pull/99/commits/aa15815474df632ac0b467ae2f0a3d6d00dcdf49

References

availproject/avail-core/core/src/constants.rs#L13
availproject/avail-core/kate/src/com.rs#L288
availproject/avail-core/kate/src/com.rs#L360-L361
availproject/avail-core/kate/src/com.rs#L435
availproject/avail-core/kate/recovery/src/commitments.rs#L116
availproject/avail-core/kate/recovery/src/com.rs#L580
availproject/avail-core/kate/recovery/src/config.rs#L1
availproject/avail-core/kate/src/gridgen/mod.rs#L277

7.4 STYLISTIC IMPROVEMENTS

// INFORMATIONAL

Description

Within the `avail-core` codebase, several code stylistic optimizations were identified.

Code Location

Inside the `core/src/asdr.rs` file, the `malformed_opaque` function is returning the result of a let binding from a block which can be simplified to returning the expression directly:

```
734 fn malformed_opaque() -> OpaqueExtrinsic {
735     use core::mem::transmute;
736
737     let op = unsigned_to_opaque();
738     let new_op = unsafe {
739         // Using `transmute` because `OpaqueExtrinsic.0` is not public.
740         let mut raw = transmute::<OpaqueExtrinsic, Vec<u8>>(op);
741         raw.pop();
742         transmute::<Vec<u8>, OpaqueExtrinsic>(raw)
743     };
744     new_op
745 }
```

For further information visit: https://rust-lang.github.io/rust-clippy/master/index.html#let_and_return

The `kate/src/com.rs` file contains the following redundant closure:

```
1151 extrinsics
1152     .into_iter()
1153     .flat_map(|data| pad_iec_9797_1(data))
1154     .map(|chunk| pad_to_chunk(chunk, chunk_size).len())
1155     .sum::<usize>()
1156     .saturated_into()
```

For further information visit: https://rust-lang.github.io/rust-clippy/master/index.html#redundant_closure

Score

Impact: 1

Likelihood: 1

Recommendation

Consider implementing the following changes:

```
734 fn malformed_opaque() -> OpaqueExtrinsic {
735     use core::mem::transmute;
736
737     let op = unsigned_to_opaque();
738     unsafe {
739         // Using `transmute` because `OpaqueExtrinsic.0` is not public.
740         let mut raw = transmute::<OpaqueExtrinsic, Vec<u8>>(op);
741         raw.pop();
742         transmute::<Vec<u8>, OpaqueExtrinsic>(raw)
743     }
744 }
```

```
1151 extrinsics
1152         .into_iter()
1153     .flat_map(pad_iec_9797_1)
1154     .map(|chunk| pad_to_chunk(chunk, chunk_size).len())
1155     .sum::<usize>()
1156         .saturated_into()
```

Remediation Plan

SOLVED: The **Avail team** solved this issue by implementing the recommended changes.

Remediation Hash

https://github.com/availproject/avail-core/pull/99/commits/aa15815474df632ac0b467ae2f0a3d6d00dcdf49

References

availproject/avail-core/core/src/asdr.rs#L734
availproject/avail-core/kate/src/com.rs#L1153

7.5 GAS OPTIMIZATIONS

// INFORMATIONAL

Description

Throughout the `avail-core` codebase, several gas optimization opportunities were identified.

Code Location

The `core/src/data_lookup/mod.rs` file contains the following snippet where it is assigning a range to the `last_range` variable then recomputes the same range instead of supplying the `last_range` variable.

```
144 let last_range = offset..compacted.size;
145 if !last_range.is_empty() {
146     index.push((prev_id, offset..compacted.size));
147 }
```

The `core/src/constants.rs` file includes assignments where constants are being assigned values derived from division operations. To enhance efficiency, it's advisable to precompute these values rather than performing unnecessary computations during runtime.

```
21 /// Cents of AVAIL has 16 decimal positions (100 Cents = $1)
22 /// 1 DOLLARS = `10_000_000_000_000_000`
23 pub const CENTS: Balance = AVAIL / 100;
24
25 /// Millicent of AVAIL has 13 decimal positions( 100 mCents = 1 cent).
26 pub const MILLICENTS: Balance = CENTS / 1_000;
27
28 /// `MILLI_AVAIL` has 15 decimal positions
29 pub const MILLI_AVAIL: Balance = AVAIL / 1_000;
30
31 /// `MICRO_AVAIL` has 12 decimal positions
32 pub const MICRO_AVAIL: Balance = MILLI_AVAIL / 1_000;
33
34 /// `NANO_AVAIL` has 9 decimal positions
35 pub const NANO_AVAIL: Balance = MICRO_AVAIL / 1_000;
36
37 /// `PICO_AVAIL` has 6 decimal positions
38 pub const PICO_AVAIL: Balance = NANO_AVAIL / 1_000;
```

Score

Impact: 1

Likelihood: 1

Recommendation

Consider implementing the following changes:

- use the `last_range` variable instead of recomputing the same range:

```
index.push((prev_id, last_range));
```

- precompute the mentioned values rather than executing unnecessary computations during runtime. For instance, given that CENTS has 16 decimals, assign it the value `10_000_000_000_000_000` instead of `AVAIL / 100`.

Remediation Plan

ACKNOWLEDGED: The **Avail team** acknowledged this issue but decided that it is not needed.

References

availproject/avail-core/core/src/data_lookup/mod.rs#L146
availproject/avail-core/core/src/constants.rs#L23 https://github.com/availproject/avail-core/blob/4b9888bd46bef404fe9b256c00811fcfcbf6d787/core/src/constants.rs
availproject/avail-core/core/src/constants.rs#L29
availproject/avail-core/core/src/constants.rs#L32
availproject/avail-core/core/src/constants.rs#L35
availproject/avail-core/core/src/constants.rs#L38

7.6 ERRONEOUS COMMENTS

// INFORMATIONAL

Description

Incorporating comments into the codebase is essential for elucidating key aspects and facilitating comprehension of the developer's intentions by others. Nevertheless, several errors were identified within these comments.

Code Location

The `core/src/constants.rs` file features several erroneous comments.
The term "DOLLARS" was erroneously used instead of "CENTS".

```
21 | /// Cents of AVAIL has 16 decimal positions (100 Cents = $1)
22 | /// 1 DOLLARS = 10_000_000_000_000_000
23 | pub const CENTS: Balance = AVAIL / 100;
```

`100` was erroneously written instead of `1000`, as 1000 millicents equal 1 cent.

```
25 | /// Millicent of AVAIL has 13 decimal positions( 100 mCents = 1 cent).
26 | pub const MILLICENTS: Balance = CENTS / 1_000;
```

Score

Impact: 1

Likelihood: 1

Recommendation

Consider implementing the following changes:

```
/// 1 CENTS = 10_000_000_000_000_000
/// Millicent of AVAIL has 13 decimal positions( 1000 mCents = 1 cent).
```

Remediation Plan

PARTIALLY SOLVED: The **Avail team** solved the first issue by removing the incorrect comment. However, the second issue remains unresolved.

Remediation Hash

https://github.com/availproject/avail-core/pull/99/commits/aa15815474df632ac0b467ae2f0a3d6d00dcdf49

References

availproject/avail-core/core/src/constants.rs#L22
availproject/avail-core/core/src/constants.rs#L25

7.7 TYPOS IN COMMENTS

// INFORMATIONAL

Description

Throughout the `avail-core` codebase, several typos were identified.

Code Location

The `core/src/asdr.rs` file contains the following typo:

```
58 | /// The SingaturePayload of UncheckedExtrinsic.
```

The `core/src/data_lookup/mod.rs` file contains the following typo:

```
191 | #[test_case( vec![(1, 10), (0, 2)] => Err(Error::DataNotSorted); "Unsorte
```

The `core/src/data_proof.rs` file contains the following typo:

```
15 | /// Max data supported on bidge (Ethereum calldata limits)
```

The `core/src/lib.rs` file contains the following typo:

```
181 | /// Calculates the Kecck 256 of arguments with NO extra allocations to jo
```

The `kate/src/com.rs` file contains the following typos:

```
94 | ExtendedGridDomianSizeInvalid(usize),
```

```
111 | /// We cannot derive PartialEq becasue PlonkError does not support it in
112 | /// and we only need to double check its discriminat for testing.
```

```
197 | // SAFETY: chunk_size comes from NonZeroU32::get(...) so we can safetly u
```

```
637 | const TCHUNK: NonZeroU32 = unsafe { NonZeroU32::new_unchecked(32) };
```

```
913 | fn verify_commitmnets_missing_row(ref xts in app_extrinsics_strategy()) {
```

The `kate/recovery/src/couscous.rs` file contains the following typos:

```
6 | .expect("Deserialising of public parameters should work for serialised pp
```

The `kate/recovery/src/matrix.rs` file contains the following typos:

```
55 | /// Refrence in format `block_number:column_number:row_number`
```

```
80 | /// Refrence in format `block_number:row_number`
```

```
147 | /// undefined behaviour if any parameter is zero.
```

The `kate/src/gridgen/mod.rs` file contains the following typos:

```
87 | // Convert each grup of extrinsics into scalars
```

```
395 | /// Dimensions of the multiproof grid. These are guarenteed to cleanly di
```

Score

Impact: 1

Likelihood: 1

Recommendation

It is advisable to rectify the mentioned typos as follows:

- `SingaturePayload` -> `SignaturePayload`
- Unsortend -> Unsorted
- bidge -> bridge
- Kecck -> Keccak
- ExtendedGridDomianSizeInvalid(usize) -> ExtendedGridDomainSizeInvalid(usize)
- becasue -> because
- discriminat -> discriminant
- safetly -> safely
- TCHUNK -> CHUNK
- verify_commitmnets_missing_row -> verify_commitments_missing_row
- Deserialising -> Deserializing
- serialised -> serialized
- Refrence -> Reference
- behaviour -> behavior
- grup -> group
- guarenteed -> guaranteed

Remediation Plan

PARTIALLY SOLVED: The **Avail team** solved most of the issues, the only ones that remain unresolved are the following:

- safetly -> safely
- TCHUNK -> CHUNK

Remediation Hash

https://github.com/availproject/avail-core/pull/99/commits/aa15815474df632ac0b467ae2f0a3d6d00dcdf49

References

availproject/avail-core/core/src/asdr.rs#L58
availproject/avail-core/core/src/data_lookup/mod.rs#L191
availproject/avail-core/core/src/data_proof.rs#L15
availproject/avail-core/core/src/lib.rs#L181
availproject/avail-core/kate/src/com.rs#L94
availproject/avail-core/kate/src/com.rs#L111-L112
availproject/avail-core/kate/src/com.rs#L197
availproject/avail-core/kate/src/com.rs#L637
availproject/avail-core/kate/src/com.rs#L913
availproject/avail-core/kate/recovery/src/couscous.rs#L6
availproject/avail-core/kate/recovery/src/matrix.rs#L55
availproject/avail-core/kate/recovery/src/matrix.rs#L80
availproject/avail-core/kate/recovery/src/matrix.rs#L147
availproject/avail-core/kate/src/gridgen/mod.rs#L87
availproject/avail-core/kate/src/gridgen/mod.rs#L395

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.

ID	PACKAGE	SHORT DESCRIPTION
RUSTSEC-2021-0139	ansi_term 0.12.1	ansi_term is Unmaintained
RUSTSEC-2020-0168	mach 0.3.2	mach is unmaintained
RUSTSEC-2022-0061	parity-wasm 0.45.0	Crate `parity-wasm` deprecated by the author
RUSTSEC-2021-0145	atty 0.2.14	Potential unaligned read

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.