HACKEN

NEAR SECURITY ANALYSIS

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Intro

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another party. Any subsequent publication of this report shall be without mandatory consent.

Name	NEAR
Website	https://near.org/
Repository	https://github.com/near/nearcore
Commit	1e781bcccfaeb9a4bb9531155193a459257afd8d
Platform	L1
Network	NEAR
Languages	Rust
Methodology	Blockchain Protocol and Security Analysis Methodology
Lead Auditor	Yaroslav Bratashchuk (y.bratashchuk@hacken.io)
Auditor	Michal Bajor (m.bajor@hacken.io) Noah Jelich (n.jelich@hacken.io) Oleksii Haponiuk (o.haponiuk@hacken.io)
Supervisor	Bartosz Barwikowski (b.barwikowski@hacken.io)
Approver	Luciano Ciattaglia (l.ciattaglia@hacken.io)
Timeline	17.07.2023 - 23.10.2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Summary

The Near Protocol, a decentralized application platform designed for scalable and user-friendly apps, has been gaining traction in the blockchain community. With a focus on usability and scalability, Near provides developers with tools to create efficient decentralized applications.

At Hacken, we conducted security research on `nearcore`, revealing findings ranging from informational to low severity.

Our primary focus was on identifying critical vulnerabilities that could potentially lead to a loss of funds or unauthorized minting of tokens, as well as vulnerabilities that could incapacitate the network or a segment of it.

The logical state of the blockchain in `nearcore` is split into two components: chain and runtime. These two are the main components in our scope of research. The chain is responsible for block and chunk production and processing, consensus, and validator selection. The runtime is responsible for applying transactions to the state.

We ensured block and chunk production and validation logic for safety, liveness, and correctness. Continuous fuzzing for block and chunk production and serialization didn't yield any issues, only a few false positives and an issue in Arbitrary derivation, which is not part of the scope of of our research. We manually explored the related codebase and tests to learn how it works. We didn't find a way to produce an invalid chunk and include it in the block. The chunk creation and distribution logic is well-designed, having undergone many refactors over the last few years. The use of erasure coding allows only a subset of validators to reconstruct an entire chunk, ensuring data integrity and availability. After checking how incoming chunks are processed and validated, we didn't find an option to corrupt a chunk that would go unnoticed.

In our deep dive into chunk production, we focused on transaction validation and processing. It's crucial to ensure the integrity and accuracy of all related processes. We set up continuous fuzzing for transaction and receipt serialization and also checked related bug bounty findings. There is one very interesting finding that exposes a vulnerability, enabling a hacker to mint tokens from thin air by duplicating receipts. This critical issue was promptly fixed the day after the bug submission and is currently being evaluated for a payout, which is anticipated to be a significant amount, acknowledging the seriousness of the vulnerability. Post-fix, we confirmed that the issue could no longer be reproduced, ensuring the robustness of our system.

Nearcore has an impressive stateless runtime implementation, but it demands a thorough understanding of its design and the mechanisms invented for specific scenarios. We like that it's possible to bundle many actions into one transaction, ensuring they all execute or fail together. As the Near Protocol is a sharded blockchain, it has developed a way to process transactions that go beyond its signer shard. This area seemed ripe for potential issues, but our investigations found it to be robust. We believe, however, that this area still requires vigilant attention from developers due to its potential impact on network economics.

Near supports nine different transaction types. The most significant one allows the execution of smart contracts inside the Near VM. Our approach here involved continuous fuzzing, aiming to identify crashes and subsequently investigate with test code. Most crashes were false positives, but some highlighted gaps in the singlepass compiler. One particular crash could disrupt the contract compilation process, but this has been addressed in the current protocol version.

The blockchain state in `nearcore` adopts an MPT structure, akin to Ethereum's but with unique modifications. While documentation on this storage approach is sparse, our codebase investigation found its design for recording and committing changes to be sound. We did identify a minor issue with TrieKey serialization related to data separator inconsistency, but this impacts only code quality, not security.

Documentation quality

Near boasts comprehensive documentation, illuminating the concepts and architecture of the node and protocol. Additionally, the specifications for all protocol components are of high quality. While we initially found that the runtime crates' documentation was outdated, this has since been updated. Additionally, the previously identified broken links throughout the codebase and readme files have been rectified. These improvements have effectively resolved the minor inconsistencies that could have potentially misled readers or developers.

The total Documentation Quality score is **10** out of 10.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Code quality

The `nearcore` is renowned for its high code quality, effectively utilizing the capabilities of the Rust programming language and its architectural patterns. As part of their ongoing commitment to security and excellence, the project developers have implemented a significant enhancement in their continuous integration (CI) process. By integrating cargo-audit into their CI system, the Near Protocol team ensures the enforcement of a "no vulnerable dependency" policy, not just as a one-time fix but as a sustained, long-term change. This proactive approach continuously safeguards against potential vulnerabilities. Furthermore, the few vulnerable or outdated external dependencies previously identified have been promptly updated. The Wasmtime dependency has been upgraded to ensure compatibility with its latest version. Concerning the singlepass compiler's validation process during compilation, a low-severity issue, it's noteworthy that a separate validation mechanism within `near_vm_runner` filters out unsupported features before compilation. The initial concerns regarding the test code quality, specifically adherence to best practices and inconsistent test coverage measurements, particularly for external tests, have been addressed and are now marked as resolved by the Near Protocol team.

The total Code Quality score is **10** out of 10.

Architecture quality

The architecture of `nearcore` is commendable. Designed with sharding at its core, it embodies scalability from day one. While no system is flawless, our comprehensive research did not identify any high-level architectural issues in `nearcore`.

The architecture quality score is **10** out of 10.

Security score

Our exhaustive research did not unearth any critical security flaws within the audit's scope. However, a few high-severity issues identified by bug hunters were promptly addressed and rectified. At this project stage, we believe that a bug bounty program is invaluable, often leading to in-depth, focused investigations.

In conclusion, `nearcore` is a high-quality blockchain project.

The security score is **10** out of 10.

Total score

Considering all metrics, the total score of the report is **10.0** out of 10.

Findings count and definitions

Severity	Findings
Critical	0
High	0
Medium	0
Low	2
Total	2

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Scope of the audit

Protocol Audit

Accounts

- Accounts implementation review
- Security vectors analysis (data availability, nonce,..)

Chain

- Tx and receipt implementation review (defaults, timestamps, assembly)
- Block and chunk production and validation logic review
- Bootstrap review (genesis, seed peers)
- Mempool review (defaults, timestamps)
- Economics and staking model review
- Standard attacks review (replay, malleability,...)

Consensus

- Consensus implementation review (validation, fork, ...)
- Attack scenarios analysis (liveness, finality, eclipse, double spend,...)
- Upgrade mechanisms review

Runtime/VM

- Runtime implementation review
- VM implementation review
- Smart contract implementation review
- Known VM Vulnerabilities review
- Attack scenarios analysis (Gas, race, stack, DoS, state implosion...)
- Contract storage implementation review

Light client integration

- Light client block validation logic review
- Light client execution proof verification logic review

Implementation

Code Quality

- Static Code Analysis
- Dynamic Code Analysis
- Tests coverage

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Protocol Tests

Node Tests

- Environment Setup
- Integration tests
- Consensus tests
- E2E transaction tests

Fuzz Tests

- Consensus fuzz tests
- Chain fuzz tests
- Runtime/VM fuzz tests

Issues

Singlepass Compiler Vulnerability: Absence of wasm Feature Validation

The `near-vm-compiler-singlepass` serves as one of the primary compilers for NearVM. However, a notable gap in its architecture is the lack of wasm feature validation prior to the actual compilation.

ID	NEAR-101
Scope	Compiler, VM
Severity	LOW
Status	Addressed in the issue

Description

The Singlepass compiler currently does not conduct a thorough validation of wasm features before initiating the compilation. This issue is further compounded by the presence of `todo!` macros within the `ArgumentRegisterAllocator` implementation, as seen here. Consequently, the following snippet of Rust code can disrupt the compilation process:

```
// both funcref and externref are falling under todo! macro case
let wasm_bytes = wat2wasm(r#"
    (module
        (type $t0 (func (param funcref externref)))
        (import "" "" (func $hello (type $t0)))
    )
    "#.as_bytes(),
)?;
let compiler = Singlepass::default();
let mut store = Store::new(compiler);
let module = Module::new(&store, wasm_bytes)?;
```

This could have been a critical issue. However, a separate validation mechanism within `near_vm_runner` prevents unsupported features from reaching the compilation stage. This validation mitigates the risk, but the underlying problem remains unresolved.

Recommendations

To enhance the robustness and reliability of the Singlepass compiler, it is strongly recommended to replace the use of `todo!` macros with proper error handling mechanisms. This improvement will not only address the current problem but will also contribute to the overall stability of the compiler, particularly when dealing with unsupported wasm features.

Inconsistent TrieKey Implementation

ID	NEAR-100
Scope	Code Quality
Severity	LOW
Status	Fixed

Description

Within the TrieKey implementation, the structs and their methods for different records such as `Account`, `ContractCode`, `AccessKey`, `ReceivedData`, etc. are defined.

For records containing multiple values (e.g., `ReceivedData` contains `receiver_id` and `data_id`), the values are packed as such:

```
TrieKey::PendingDataCount { receiver_id, receipt_id } => {
  buf.push(col::PENDING_DATA_COUNT); // Column name
  buf.extend(receiver_id.as_ref().as_bytes()); // First data element
  buf.push(ACCOUNT_DATA_SEPARATOR); // Data separator
  buf.extend(receipt_id.as_ref()); // Second data element
}
```

However, there is an inconsistency for the `AccessKey` record, possibly due to a mistake. The column name identifier is used instead of the `ACCOUNT_DATA_SEPARATOR` constant. This bug is propagated in further usage.

Here:

```
TrieKey::AccessKey { account_id, public_key } => {
  col::ACCESS_KEY.len() * 2 + account_id.len() + public_key.len()
}
```

And here:

```
TrieKey::AccessKey { account_id, public_key } => {
  buf.push(col::ACCESS_KEY);
  buf.extend(account_id.as_ref().as_bytes());
  buf.push(col::ACCESS_KEY);
  buf.extend(public_key.try_to_vec().unwrap());
}
```

Due to validation elsewhere in the code and the fact that the `ACCOUNT_DATA_SEPARATOR` is the same length as the `col::ACCESS_KEY`, as well as the fact that code was written around this bug (adding the `next_element` function which takes the separator as a parameter to handle the edge case), this does not cause problems or crashes.

However, in order to maintain a high code quality, this issue must be fixed.

Further down the code, there is another inconsistency within the inner tests module. The `[test_account_id_from_trie_key](https://github.com/hknio/nearcore/blob/audit/core/primitives/src/trie_key.rs#L648)` runs only against the first item in `[OK_ACCOUNT_IDS](https://github.com/hknio/nearcore/blob/audit/core/primitives/src/trie_key.rs#L448)`:

```
fn test_account_id_from_trie_key() {
    let account_id = OK_ACCOUNT_IDS[0].parse::<AccountId>().unwrap();
    // ...
}
```

Recommendation

To address the issue in the TrieKey implementation, the following actions are recommended:

1. **Fix AccessKey Record:** Correct the usage of the `ACCOUNT_DATA_SEPARATOR` in the `AccessKey` record by replacing it with the appropriate column name identifier (`col::ACCESS_KEY`).

2. **Code Refactoring:** Review and refactor the code that relies on the incorrect usage of the `ACCOUNT_DATA_SEPARATOR` to ensure consistency and improve code quality.

3. **Comprehensive Testing:** Enhance the tests in the TrieKey implementation. Specifically, modify the `test_account_id_from_trie_key` function to iterate through the entire array of account IDs, similar to other tests. This ensures comprehensive coverage and accurate validation.

Address Broken Links Throughout Codebase

ID	NEAR-106
Scope	Documentation
Status	Fixed

Description

Multiple broken URLs have been identified within the project's codebase, documentation, and README files. These need to be reviewed and corrected to maintain the integrity and accuracy of the project documentation.

An analysis using lychee unveiled broken URLs in 22 locations, detailed below:

- `./docs/architecture/README.md`:
  - File not found: `/workspaces/nearcore/docs/architecture/style.md`
- `./docs/architecture/how/README.md`:
  - 404 Error: Nearcore Sync Code Reference
- `./docs/architecture/how/epoch.md`:
  - DNS Error for http://go/mainnet-genesis
- `./docs/architecture/gas/estimator.md`:
  - 404 Error: QEMU Documentation
- `./docs/architecture/network.md`:
  - 404 Error: Peer Manager Code Reference
  - 404 Error: Actix Documentation
- `./docs/practices/when_to_use_private_repository.md` & `./docs/practices/security_vulnerabilities.md`:
  - 404 Error: https://github.com/near/nearcore-private
- `./docs/practices/testing/README.md`:
  - Multiple 404 Errors, DNS error: Detailed list of broken URLs needs review.
- `./docs/misc/README.md`:
  - File not found: `/workspaces/nearcore/docs/misc/Cargo.toml`
- `./runtime/runtime-params-estimator/README.md`:
  - File not found: `/workspaces/nearcore/runtime/runtime-params-estimator/continuous-estimation/README.md`
- `./runtime/near-vm/api/README.md`:
  - 404 Error: Wasmer Dylib
  - 404 Error: Wasmer Staticlib
  - 404 Error: Wasmer Universal

- `./runtime/near-vm/engine-universal/README.md`:
  - 404 Error: Wasmer Example Engine Universal
- `./runtime/near-vm/engine/README.md`:
  - 404 Error: Various Wasmer URLs; please verify each link.
- `./chain/rosetta-rpc/README.md`:
  - 404 Error: Contribute to Nearcore
  - Connection refused: http://localhost:3040/api/spec
- `./chain/jsonrpc/res/debug.html`:
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/sync`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/network_info`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/last_blocks`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/validator`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/chain_n_chunk_info`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/tier1_network_info`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/pages/epoch_info`
  - File not found: `/workspaces/nearcore/chain/jsonrpc/res/debug/client_config`
- `./chain/epoch-manager/README.md`:
  - 404 Error: Validator Rewards Calculation
- `./chain/chunks/README.md`:
  - 404 Error: Near Nightshade
- `./tools/debug-ui/README.md`:
  - Connection refused: http://localhost:3000/logviz
- `./tools/state-viewer/README.md`:
  - 404 Error: Near Ops Pull Request
- `./pytest/tests/mocknet/README.md`:
  - 404 Error: Near Ops Repository
- `./venv/lib/python3.11/site-packages/deepdiff-6.5.0.dist-info/AUTHORS.md`:
  - 404 Error: Boba-2 GitHub Profile
- `./CONTRIBUTING.md`:
  - File not found: `/workspaces/nearcore/docs/protocol_upgrade.md`

Recommendations

- **Review & Correct URLs**: Each broken URL should be reviewed. Replace those that have moved with their current counterparts, and for those that no longer exist, decisions should be made on whether to remove or replace them.

- **Update Dead Files & Pages**: For pages or files that cannot be found, review whether these need to be reintroduced, updated, or if references to them should be removed from the documentation.

Taking these steps will ensure that the project's documentation remains robust, accurate, and user-friendly, facilitating smoother development and collaboration processes.

Address Vulnerable, Outdated, and Unmaintained Dependencies

ID	NEAR-107
Scope	Dependency Management, Code Quality
Status	Fixed

Description

A `cargo audit` has revealed multiple dependencies that are vulnerable, unmaintained, or outdated. Below is a detailed list:

Vulnerable Dependencies:

- **Crate: ed25519-dalek (Version: 1.0.1)**
  - RUSTSEC-2022-0093: Vulnerable to double public key signing function oracle attack. Upgrade to version >=2.0.
- **Crate: h2 (Version: 0.3.13)**
  - RUSTSEC-2023-0034: Resource exhaustion vulnerability may lead to Denial of Service (DoS). Upgrade to version >=0.3.17.
- **Crate: libsqlite3-sys (Version: 0.24.2)**
  - RUSTSEC-2022-0090: Vulnerable to CVE-2022-35737. Upgrade to version >=0.25.1.
- **Crate: openssl (Version: 0.10.48)**
  - RUSTSEC-2023-0044: Buffer over-read vulnerability in `X509VerifyParamRef::set_host`. Upgrade to version >=0.10.55.
- **Crate: remove_dir_all (Version: 0.5.3)**
  - RUSTSEC-2023-0018: Race condition enabling link following and TOCTOU. Upgrade to version >=0.8.0.
- **Crate: time (Version: 0.1.44)**
  - RUSTSEC-2020-0071: Potential segfault. Upgrade to version >=0.2.23.

Unmaintained Dependencies:

- **Crate: ansi_term (Version: 0.12.1)**
  - RUSTSEC-2021-0139: The crate is unmaintained.
- **Crate: mach (Version: 0.3.2)**
  - RUSTSEC-2020-0168: The crate is unmaintained.
- **Crate: memmap (Version: 0.7.0)**

- RUSTSEC-2020-0077: The crate is unmaintained.

- **Crate: parity-wasm (Versions: 0.41.0 and 0.42.2)**
  - RUSTSEC-2022-0061: The crate is deprecated and unmaintained.
- **Crate: serde_cbor (Version: 0.11.2)**
  - RUSTSEC-2021-0127: The crate is unmaintained.

Deprecated or Yanked Dependencies:

- **Crate: cpufeatures (Version: 0.2.2)**
  - Yanked from crates.io.
- **Crate: crossbeam-channel (Version: 0.5.4)**
  - Yanked from crates.io.
- **Crate: ed25519 (Version: 1.5.1)**
  - Yanked from crates.io.
- **Crate: hermit-abi (Version: 0.3.1)**
  - Yanked from crates.io.

Recommendations

1. **Upgrade Dependencies**: Act promptly to upgrade the listed dependencies to their recommended versions or to viable alternatives to mitigate any associated risks.
2. **Dependency Maintenance Strategy**: Develop and adhere to a maintenance strategy for dependencies to ensure the project utilizes well-supported and secure libraries.

Addressing these issues proactively will fortify the project's security posture and facilitate smoother, risk-mitigated development going forward.

Inherent risk with use of "clamp" function for gas price validation

ID	NEAR-103
Scope	Block Production, Gas Price Management
Status	Addressed

Description

The current implementation for gas price validation relies on the built-in `clamp` function, which is prone to panicking if the condition for minimum and maximum gas prices fails. While these values are constants as of now, future iterations that allow users to set their own min/max limits could introduce instability and lead to node crashes.

Recommendations

1. **Validate Genesis Config Values**: To prevent future risks, it's advisable to include initial validation for min/max gas prices in the genesis config (here). This would catch potential issues before they propagate, enhancing overall network stability.

2. **Replace Clamp with Custom Validation Logic**: It is advisable to replace the clamp function with custom validation logic that uses proper error types instead of panicking.

3. **Improve Error Handling**: Along with replacing clamp, enriching the error-handling mechanism can contribute to better system resilience and provide more informative feedback to users or administrators when anomalies occur.

NearVM runtime crates has inadequate documentation and TODO annotations

The NearVM runtime crates, which were forked from Wasmer, exhibit a notable lack of proper documentation. It is essential for the community and the developers to understand the modifications made, and the reasons behind them, especially given the nature and importance of blockchain-related projects.

ID	NEAR-102
Scope	Documentation, Runtime Crates
Status	Fixed

Description

The integrity and clarity of the NearVM runtime crates are compromised by two primary issues:

1. **Lack of Documentation on Changes from Wasmer**: There's an absence of explicit documentation highlighting the changes made after forking from Wasmer. Such documentation is imperative to understand the evolution, divergence, and unique features or modifications introduced in the NearVM version.

2. **Unexplained TODO Annotations**: Numerous `TODO` annotations are scattered throughout the crates. While some `TODOs` are generic or benign, others raise significant concern due to their alarming content. A particularly disconcerting comment reads:

   ```
   // TODO: What in damnation have you done?! – Bannon
   ```

   Annotations like these suggest that there might be sections of the code that are potentially problematic or deviate significantly from best practices.

Recommendations

1. **Document Changes from Wasmer**: Create a comprehensive documentation section that clearly outlines the differences between NearVM's runtime crates and Wasmer's original codebase. This will help developers, auditors, and users understand the reasons behind specific decisions and modifications.

2. **Review and Address TODO Annotations**: Conduct a thorough review of all `TODO` annotations in the crates. Prioritize those with alarming or ambiguous content, like the one cited above, and either address the underlying issues or provide more context to explain the annotations.

3. **Establish a Strong Documentation Practice**: To maintain clarity and transparency, implement a robust documentation strategy. Encourage contributors to document significant changes, decisions, and areas of concern. This not only fosters transparency but also aids in future development, debugging, and audits.

Rectification of Documentation Inconsistencies

ID	NEAR-105
Scope	Documentation, Tests
Status	Fixed

Description

The project's documentation, code comments, and sample commands exhibit various inconsistencies that could mislead readers or developers working with the project. Addressing these inaccuracies is crucial for maintaining the integrity and clarity of the project's documentation.

Incorrect URL for Runner Test Results:

The documentation for Testing Practices provides an incorrect link to the testing results. It reads:

> Expensive and Python tests are not part of CI and are run by a custom nightly runner. The results of the latest runs are available **here**.

However, the accurate URL for test results is https://nayduck.near.org/#/.

Non-Existent Test Target Documentation:

The documentation references a non-existent test target, `cross_shard_tx`. The Test Hierarchy Section includes a command supposedly related to this target, but no such target exists in the codebase.

```
cargo nextest run --package near-client --test cross_shard_tx tests::test_cross_shard_tx --all-features
```

Misguiding Python Tests Documentation:

The Python test documentation suggests using a plain cargo build command for local test runs. However, this guidance is insufficient as many tests require specific features to run. The documentation fails to specify which features should be paired with which tests, leading to potential confusion and misuse.

Recommendations

1. **Update Runner Test Results URL**: Replace the outdated URL in the Testing Practices documentation with the correct NayDuck URL.
2. **Remove or Correct Invalid Test Target Reference**: Either remove the `cross_shard_tx` reference from the Test Hierarchy Section or update the documentation to reference a valid test target.
3. **Provide Clear Guidance for Python Tests**: Revise the Python test documentation to include explicit instructions on which features should be used with each test, providing developers with clear and accurate guidance for running tests locally.

Test Fixtures And Coverage Analysis

ID	NEAR-108
Scope	Test Coverage, Code Quality
Status	Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Description

The test coverage was measured for Rust based tests using `cargo llvm-cov` yielding the following:

Function Coverage: 61.56% (11509/18696)

Line Coverage: 71.75% (96446/134413)

Region Coverage: 56.82% (41749/73482).

A detailed analysis is provided with the report.

Furthermore, several flaky or misconfigured tests were found:

```
FAIL [   0.203s] integration-tests tests::client::features::chunk_nodes_cache::compare_node_counts
FAIL [   0.187s] near-vm-runner tests::cache::test_wasmer2_artifact_output_stability
SIGABRT [   0.429s] near-vm-runner tests::regression_tests::memory_size_alignment_issue
SIGABRT [   5.672s] near-vm-runner tests::rs_contract::attach_unspent_gas_but_burn_all_gas
SIGABRT [   6.144s] near-vm-runner tests::rs_contract::attach_unspent_gas_but_use_all_gas
SIGABRT [   6.081s] near-vm-runner tests::rs_contract::ext_account_balance
SIGABRT [   5.254s] near-vm-runner tests::rs_contract::ext_account_id
SIGABRT [   5.816s] near-vm-runner tests::rs_contract::ext_attached_deposit
SIGABRT [   4.591s] near-vm-runner tests::rs_contract::ext_block_index
SIGABRT [   5.373s] near-vm-runner tests::rs_contract::ext_block_timestamp
SIGABRT [   4.699s] near-vm-runner tests::rs_contract::ext_predecessor_account_id
SIGABRT [   5.225s] near-vm-runner tests::rs_contract::ext_prepaid_gas
```

Recommendations

1. **Improve Test Score**: To enhance the quality and reliability of your software, consider setting a specific target for code coverage (e.g., 80% or higher), and work on creating additional test cases to achieve this goal. Increasing test coverage can help identify and prevent potential issues, leading to a more robust application..
2. **Enable External Test Coverage**: To measure coverage for external tests in your Rust project, follow these steps using the `cargo-llvm-cov` tool: configure essential environment variables, remove artifacts that may influence coverage results, build Rust binaries, execute necessary commands, and finally, generate a coverage report in LCOV format. This will provide a comprehensive analysis of your code's test coverage.
3. **Implement *llvm-cov* Into CI/CD**: Incorporating `cargo-llvm-cov` into your testing process ensures comprehensive coverage analysis, enabling you to assess the effectiveness of your test suite in identifying potential issues and vulnerabilities in your Rust codebase.
4. **Fix Failing Tests:** By fixing these issues, you will enhance the overall reliability of your test suite, leading to more accurate and dependable test results. This, in turn, will boost your confidence in the software's stability and facilitate more effective development and quality assurance processes.

Taking these steps in advance will strengthen the project's security stance and enable more seamless, risk-mitigated development in the future.

Upgrade Wasmtime Dependency and Adjust for API Changes

ID	NEAR-104
Scope	Dependency Management, Code Quality
Status	Fixed

Description

Wasmtime Dependency:

- Current version: 4.0.0
- Recommended version: 12.0.1
- Reasoning: Transitioning to version 12.0.1 will bring Near up to date with the most recent optimizations, bug fixes, and potential security enhancements from Wasmtime. However, it's noteworthy that version 12.0.1 introduces API changes (since version 6.0.0) which will necessitate code adjustments.

Recommendations

Upgrade the Wasmtime dependency from version 4.0.0 to 12.0.1. Given the altered API in version 12.0.1, the codebase should be reviewed and modified accordingly to ensure compatibility. This update will not only keep Near aligned with best development practices but also ensure the platform leverages the latest offerings from Wasmtime.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken disclaimer

The code base provided for audit has been analyzed according to the latest industry code quality, software processes and cybersecurity practices at the date of this report, with discovered security vulnerabilities and issues the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functional specifications). The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code (branch/tag/commit hash) submitted to and reviewed, so it may not be relevant to any other branch. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits, public bug bounty program and CI/CD process to ensure security and code quality. English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical disclaimer

Protocol Level Systems are deployed and executed on hardware and software underlying platforms and platform dependencies (Operating System, System Libraries, Runtime Virtual Machines, linked libraries, etc.). The platform, programming languages, and other software related to the Protocol Level System may have vulnerabilities that can lead to security issues and exploits. Thus, Consultant cannot guarantee the explicit security of the Protocol system in full execution environment stack (hardware, OS, libraries, etc.)