Quantstamp

Astar Account Unification (SS58 + H160)

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Account, Cross VM
Timeline	2023-11-15 through 2023-11-27
Language	Rust
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Design and Implementation Doc User Facing Doc
Source Code	• AstarNetwork/Astar #1110c86
Auditors	• Andy Lin Senior Auditing Engineer • Rabib Islam Auditing Engineer • Poming Lee Senior Auditing Engineer

Documentation quality	Medium
Test quality	Undetermined
Total Findings	5 Fixed: 1 Acknowledged: 4
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	1 Acknowledged: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	2 Acknowledged: 2

Summary of Findings

During the audit, we examined the `unified-accounts` pallet, which is responsible for establishing mappings between native Substrate accounts (SS58 format) and EVM accounts (H160 format). The pallet is structured with simple logic and seems to be coded effectively. It also appears that the development team has taken into account common attack factors, including signature replay and duplicated mapping, demonstrating consideration for essential security aspects.

In our audit, we have identified some issues concerning Astar's account unification feature, including risks in the account binding process, the necessity of state clearance in specific functions, and security implications of linking certain Substrate accounts. Additionally, the `cargo audit` command revealed critical vulnerabilities in dependencies, highlighting areas for security enhancements. We recommend the team fix all issues.

Fix Review Update: The team has fixed or acknowledged all issues. We would suggest the team to check with the upstream dependency team to see if AST-5 can be further mitigated/resolved despite the potential lack of immediate impact to the project.

ID	DESCRIPTION	SEVERITY	STATUS
AST-1	The Account Can Be Reaped when Binding Accounts	Medium	Fixed
AST-2	Potential Security Implications of Account Unification with Special Substrate Accounts	Low	Acknowledged
AST-3	Ensuring State Clearance/transfer Before `claim_evm_address()` Execution	Informational	Acknowledged
AST-4	Ability to Reuse Signature with Reaped and Revived Accounts	Informational	Acknowledged
AST-5	Critical Vulnerabilities and Unmaintained Packages in Cargo Audit Report	Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Block timing and ordering dependence
- Arithmetic overflow / underflow
- Storage efficiency and bloating
- Unsafe inter-pallet calls
- Consensus mechanism vulnerabilities
- Mismanagement of session keys
- Cross-pallet interaction vulnerabilities
- Resource exhaustion / Denial of Service (DoS)
- Access control and permissions flaws
- Power centralization in governance mechanisms
- Inconsistencies with on-chain business logic
- Redundant code and logic duplication
- Weight and resource consumption management
- Insecure on-chain randomness usage

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The scope of this audit is limited to the unified accounts feature.

Files Included

- `/pallets/unified-accounts`
- `UnifiedAddressMapper` trait in `primitives/src/evm.rs` (see: code).

Findings

AST-1 The Account Can Be Reaped when Binding Accounts

Medium Fixed

> **Update**
> The team fixed the issue in `434fa95` as recommended.

File(s) affected: `lib.rs`

Description: During the account binding process, which encompasses `claim_evm_address()` and `claim_default_evm_address()`, the implementation incurs a storage fee through the `charge_storage_fee()` function. This fee charging mitigates DoS risks and ensures payment for the storage associated with account binding. However, the `charge_storage_fee()` function calls `T::Currency::burn_from()`, which poses a risk of reducing the native tokens below the Existential Deposit (ED). The `burn_from()` function's implementation (see: code) verifies the balance using `reducible_balance(who, Expendable, force)`, which does not protect against account reaping.

For most users, the likelihood and impact are low because the Existential Deposit (ED) is configured to be quite low (1,000,000, which is 0.000001 when considering 18 decimal digits) in terms of real value. However, since the XVM extension also automatically triggers `claim_default_evm_address()`, there is a risk that the WASM contract account could be accidentally reaped, leading to the deletion of the on-chain contract code and potentially having a higher impact. Therefore, we consider the severity of this issue to be medium, rather than low.

Recommendation: It is advisable to add a check on the amount to be burned using `reducible_balance()` with `Preserve` as the preservation status. This check should be conducted before calling `burn_from()` to prevent the account from being reaped or reduced to dust.

AST-2
Potential Security Implications of Account Unification with Special Substrate Accounts

Low | Acknowledged

Update

The team acknowledged the issue with the following statement:

> The cases described in above situation are not explicitly a security issue in the design of AU and possibility of happening is quite low. In all such cases user (a multi-sig account or wasm contract) it should be quite obvious to users that this will lead to grant sole control over the special account

Description: The account unification feature in Astar, which allows binding an Ethereum Virtual Machine (EVM) Externally Owned Account (EOA) to a Substrate account, presents potential security risks when linked with certain types of Substrate accounts. Specifically, there are concerns regarding the interaction between this feature and special Substrate accounts like multisig accounts and Wasm contract accounts.

1. **Multisig Accounts**: If a multisig Substrate account, which typically requires multiple signatories for transactions, is mapped to a single EVM EOA, this could inadvertently grant sole control over the multisig account's balance to the EVM EOA. This undermines the multisig account's security model that relies on multiple approvals.
2. **WASM Contract Accounts**: Our audit indicates that currently, it is not feasible to bind a WASM contract account to an EVM EOA. This is due to the WASM contract's inability to arbitrarily call any pallets, thereby preventing the `claim_evm_address()` call. However, future code changes in Astar should remain vigilant about this aspect. If such a binding were to become possible, it would enable an EOA to directly transfer balances held by the contract, potentially creating a security loophole.

Recommendation: We do not have an immediate suggestion as there doesn't seem to be a straightforward solution, and both the impact and likelihood are low. However, the team should keep this in mind when developing new features and ensure it is documented.

AST-3
Ensuring State Clearance/transfer Before `claim_evm_address()` Execution

Informational | Acknowledged

Update

The team acknowledged the issue with the following statement:

> The scenario is possible and we have updated the developer docs to add this. Since XVM is still under testing on our testnet and not available on mainnet, there is no potential of loss of funds as of now. If/When XVM is launched on mainnet we will prepare ample amount of documentation to let users/developers aware of this scenario. Docs Updated here - https://github.com/AstarNetwork/astar-docs/commit/665be20ea13ae72a2b95d09d250c023ce1f2c98d

File(s) affected: `lib.rs`,

Description: The function `claim_evm_address()` assists users in transferring their native token to a newly assigned SS58 address from the default-mapping address. While migration for XC20 tokens at the frontend is in place, users still need to clear other states related to the default-mapping address. This is especially critical with the XVM feature, where an EVM address may already be linked to an active default SS58 address within the WASM contracts. If `claim_evm_address()` is invoked without first clearing these states, there's a risk that they might become inaccessible as the EVM address is remapped to a new SS58 address, disrupting its access via XVM.

Exploit Scenario: here is a sample scenario:

1. A user possesses ERC20 tokens within a Wasm contract VM, controlled by their default-mapping EVM account.
2. The user initiates `claim_evm_address()` to bind their EVM address to a new SS58 address, unaware of the need to transfer ERC20 tokens beforehand.
3. The function successfully remaps the EVM address to the new SS58 address but does not transfer the ERC20 tokens from the Wasm contract VM linked to the original EVM address.
4. As a result, the ERC20 tokens remain held by the old SS58 address that uses the default mapping algorithm from an EVM address. Since the tokens weren't transferred prior to the remapping, they became inaccessible under the new SS58 address.
5. This leads to the user losing access to the ERC20 tokens, causing potential asset loss or operational issues in the Wasm contract VM, as the tokens are now stranded under the original EVM address.

Recommendation: Enhance user guidance and warnings for `claim_evm_address()`. Inform users about the need to manually clear assets and states associated with the default-mapping address. This can be done through updated documentation, user interface prompts, and function-specific warnings, emphasizing the manual transfer of non-native tokens and related states.

AST-4
Ability to Reuse Signature with Reaped and Revived Accounts

Informational | Acknowledged

Update

The team acknowledged the issue with the following reasons provided (rephrased a bit from their original words in our Slack discussion):

1. If the accounts were connected before that means both the EVM and Native wallet in context are owned by the same user.
2. In the event of reaping (mapping erased) and afterward, both the EVM and Native wallets are still owned by the same user; thus the above scenario is unlikely as in my understanding it assumes EVM and Native wallets are controlled by different users.

In the context of AU it is more acceptable since the signature provider (EVM wallet) and signature consumer (Native wallet) are controlled by the same entity.

File(s) affected: `lib.rs`

Description: After the process of account reaping, the mapping data between `NativeToEvm` and `EvmToNative` is erased. This action theoretically allows an EVM (Ethereum Virtual Machine) address to link to a new Substrate address through a fresh signature. However, the original Substrate address retains the ability to replay the EVM signature from the previous mapping. This replay can interfere with and potentially prevent the establishment of a new link to a different address.

This issue is classified as informational because the replay of the signature can only link back to the original Substrate account, which is likely to be the intended case in most, if not all, scenarios. Nonetheless, this vulnerability may lead to unexpected outcomes that do not align with user intentions.

Exploit Scenario: Here is a sample scenario:

1. Alice originally linked her EVM address to her substrate account: `s1`.
2. One day, the private key of the substrate account `s1` is leaked.
3. Alice wants to save her EVM address by enforcing an account reaping and attempting to re-link to a new substrate account `s2`.
4. The attacker holding the leaked `s1` notices that, and front-runs the call to enforce linking back to the `s1` account instead of `s2`.

Recommendation: While this behavior is not inherently problematic, as the SS58 address was originally mapped to the EVM address, it is recommended to introduce a validity deadline for signatures. This change would prevent the long-term reuse of signatures and reduce the chance of unforeseen impacts in the future.

AST-5
Critical Vulnerabilities and Unmaintained Packages in Cargo Audit Report

Undetermined | Acknowledged

Alert

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The packages listed in the findings are transient dependencies from Polkadot SDK and are not exclusive to the AU pallet itself and it is out of our control to upgrade them.

We agree that these packages might not be totally in control of Astar but we would like to suggest they start a discussion with the Polkadot SDK team to resolve/mitigate this issue.

File(s) affected: `Cargo.toml`, `Cargo.lock`

Description: The `cargo audit` report highlights potential vulnerabilities in dependencies, which are noteworthy. Although most are unrelated to the code changes and the main focus of this audit, addressing these dependency issues is recommended. Below is a summary of the report:

1. **Double Public Key Signing Function Oracle Attack on** `ed25519-dalek` (RUSTSEC-2022-0093):
   - **Severity**: High. This vulnerability allows an adversary to potentially extract the private key due to unsafe API design in `ed25519-dalek` versions prior to 2.0.
   - **Affected Version**: 1.0.1.
   - **Recommendation**: Upgrade to version 2.0 or higher, which has revised public APIs to avoid this issue.
2. **Potential Segfault in the** `time` **crate** (RUSTSEC-2020-0071):

   - **Severity**: High. This issue can cause a segmentation fault on Unix-like operating systems, potentially leading to code execution or memory corruption.
   - **Affected Version**: 0.1.45.
   - **Recommendation**: Upgrade to version 0.2.23 or higher, or switch to the 0.3 series of the `time` crate.
3. **CPU Denial of Service in Certificate Path Building in `webpki`** (RUSTSEC-2023-0052):
   - **Severity**: High. This vulnerability can lead to a CPU denial-of-service attack when processing a pathological certificate chain.
   - **Affected Versions**: 0.21.4 and 0.22.0.
   - **Recommendation**: Upgrade to version 0.22.2 or higher.

Additionally, several unmaintained and yanked packages are listed in the warnings section, such as `aes-soft`, `aesni`, `ansi_term`, `mach`, and `parity-wasm`. These do not pose immediate security threats but can become problematic due to a lack of updates and support. It is recommended to replace them with maintained alternatives where possible.

Recommendation: Immediate action is advised to address the critical vulnerabilities, especially in the `ed25519-dalek`, `time`, and `webpki` crates. Moreover, the unmaintained packages should be reviewed for potential replacement with more actively maintained alternatives to ensure long-term stability and security.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. On `lib.rs#L31` of the unified-account pallet, there is a typo on the comment `connect their evm address to have a unified experence across the different VMs`. The word `experence` should be `experience`.
2. On `lib.rs#L370` of the unified-account pallet, there is a typo on the comment `OnKilledAccout hooks implementation` on top of the line `impl<T: Config> OnKilledAccount...`. The word `OnKilledAccout` should be `OnKilledAccount`.
3. On `lib::charge_storage_fee()#L272` of the unified-account pallet, there is a typo on the comment `Charge the (exact) storage fee (politely)`. The word `politely` should be `politely`.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Cargo Audit 0.18.3

Steps taken to run the tools:
- Installed via `cargo install cargo-audit`
- Ran `cargo audit --json`

Automated Analysis

Cargo Audit

{ "database": { "advisory-count": 578, "last-commit": "3338fcfb59cea5fcd7d2a4e7fe24cbc7cb778003", "last-updated": "2023-11-11T14:59:01Z" }, "lockfile": { "dependency-count": 1234 }, "settings": { "target_arch": null, "target_os": null, "severity": null, "ignore": [], "informational_warnings": ["unmaintained", "unsound", "notice"] }, "vulnerabilities": { "found": true, "count": 4, "list": [{ "advisory": { "id": "RUSTSEC-2022-0093", "package":

"ed25519-dalek", "title": "Double Public Key Signing Function Oracle Attack on `ed25519-dalek`", "description": "Versions of `ed25519-dalek` prior to v2.0 model private and public keys as\nseparate types which can be assembled into a `Keypair`, and also provide APIs\nfor serializing and deserializing 64-byte private/public keypairs.\n\nSuch APIs and serializations are inherently unsafe as the public key is one of\nthe inputs used in the deterministic computation of the `S` part of the signature,\nbut not in the `R` value. An adversary could somehow use the signing function as\nan oracle that allows arbitrary public keys as input can obtain two signatures\nfor the same message sharing the same `R` and only differ on the `S` part.\n\nUnfortunately, when this happens, one can easily extract the private key.\n\nRevised public APIs in v2.0 of `ed25519-dalek` do NOT allow a decoupled\nprivate/public keypair as signing input, except as part of specially labeled\n\"hazmat\" APIs which are clearly labeled as being dangerous if misused.", "date": "2022-06-11", "aliases": ["GHSA-w5vr-6qhr-36cc"], "related": [], "collection": "crates", "categories": ["crypto-failure"], "keywords": [], "cvss": null, "informational": null, "references": [], "source": null, "url": "https://github.com/MystenLabs/ed25519-unsafe-libs", "withdrawn": null, "license": "CC0-1.0" }, "versions": { "patched": [">=2"], "unaffected": [] }, "affected": null, "package": { "name": "ed25519-dalek", "version": "1.0.1", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "c762bae6dcaf24c4c84667b8579785430908723d5c889f469d76a41d59cc7a9d", "dependencies": [{ "name": "curve25519-dalek", "version": "3.2.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "ed25519", "version": "1.5.3", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "rand", "version": "0.7.3", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "serde", "version": "1.0.171", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "sha2", "version": "0.9.9", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "zeroize", "version": "1.6.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null } }, { "advisory": { "id": "RUSTSEC-2020-0071", "package": "time", "title": "Potential segfault in the time crate", "description": "### Impact\n\nUnix-like operating systems may segfault due to dereferencing a dangling pointer in specific circumstances. This requires an environment variable to be set in a different thread than the affected functions. This may occur without the user's knowledge, notably in a third-party library.\n\nThe affected functions from time 0.2.7 through 0.2.22 are:\n\n- `time::UtcOffset::local_offset_at`\n- `time::UtcOffset::try_local_offset_at`\n- `time::UtcOffset::current_local_offset`\n- `time::UtcOffset::try_current_local_offset`\n- `time::OffsetDateTime::now_local`\n- `time::OffsetDateTime::try_now_local`\n\nThe affected functions in time 0.1 (all versions) are:\n\n- `at`\n- `at_utc`\n- `now`\n\nNon-Unix targets (including Windows and wasm) are unaffected.\n\n### Patches\n\nPending a proper fix, the internal method that determines the local offset has been modified to always return `None` on the affected operating systems. This has the effect of returning an `Err` on the `try_*` methods and `UTC` on the non-`try_*` methods.\n\nUsers and library authors with time in their dependency tree should perform `cargo update`, which will pull in the updated, unaffected code.\n\nUsers of time 0.1 do not have a patch and should upgrade to an unaffected version: time 0.2.23 or greater or the 0.3 series.\n\n### Workarounds\n\nA possible workaround for crates affected through the transitive dependency in `chrono`, is to avoid using the default `oldtime` feature dependency of the `chrono` crate by disabling its `default-features` and manually specifying the required features instead.\n\n#### Examples:\n\n`Cargo.toml`:\n\n`toml\nchrono = { version = \"0.4\", default-features = false, features = [\"serde\"] }\n`\n\n`toml\nchrono = { version = \"0.4.22\", default-features = false, features = [\"clock\"] }\n`\n\nCommandline:\n\n`bash\ncargo add chrono --no-default-features -F clock\n`\n\nSources:\n- chronotope/chrono#602 (comment)\n- vityafx/serde-aux#21", "date": "2020-11-18", "aliases": ["CVE-2020-26235", "GHSA-wcg3-cvx6-7396"], "related": [], "collection": "crates", "categories": ["code-execution", "memory-corruption"], "keywords": ["segfault"], "cvss": "CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H", "informational": null, "references": [], "source": null, "url": "https://github.com/time-rs/time/issues/293", "withdrawn": null, "license": "CC0-1.0" }, "versions": { "patched": [">=0.2.23"], "unaffected": ["=0.2.0", "=0.2.1", "=0.2.2", "=0.2.3", "=0.2.4", "=0.2.5", "=0.2.6"] }, "affected": { "arch": [], "os": ["linux", "redox", "solaris", "android", "ios", "macos", "netbsd", "openbsd", "freebsd"], "functions": { "time::OffsetDateTime::now_local": ["<0.2.23"], "time::OffsetDateTime::try_now_local": ["<0.2.23"], "time::UtcOffset::current_local_offset": ["<0.2.23"], "time::UtcOffset::local_offset_at": ["<0.2.23"], "time::UtcOffset::try_current_local_offset": ["<0.2.23"], "time::UtcOffset::try_local_offset_at": ["<0.2.23"], "time::at": ["^0.1"], "time::at_utc": ["^0.1"], "time::now": ["^0.1"] } }, "package": { "name": "time", "version": "0.1.45", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "1b797afad3f312d1c66a56d11d0316f916356d11bd158fbc6ca6389ff6bf805a", "dependencies": [{ "name": "libc", "version": "0.2.147", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "wasi", "version": "0.10.0+wasi-snapshot-preview1", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "winapi", "version": "0.3.9", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null } }, { "advisory": { "id": "RUSTSEC-2023-0052", "package": "webpki", "title": "webpki: CPU denial of service in certificate path building", "description": "When this crate is given a pathological certificate chain to validate, it will\nspend CPU time exponential with the number of candidate certificates at each\nstep of path building.\n\nBoth TLS clients and TLS servers that accept client certificate are affected.\n\nThis was previously reported in\nhttps://github.com/briansmith/webpki/issues/69 and re-reported recently\nby Luke Malinowski.\n\nwebpki 0.22.1 included a partial fix and webpki 0.22.2 added further fixes.", "date": "2023-08-22", "aliases": ["GHSA-8qv2-5vq6-g2g7"], "related": ["CVE-2018-16875"], "collection": "crates", "categories": ["denial-of-service"], "keywords": ["certificate", "path building", "x509"], "cvss": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H", "informational": null, "references": [], "source": null, "url": null, "withdrawn": null, "license": "CC0-1.0" }, "versions": { "patched": [">=0.22.2"], "unaffected": [] }, "affected": null, "package": { "name": "webpki", "version": "0.21.4", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "b8e38c0608262c46d4a56202ebabdeb094cef7e560ca7a226c6bf055188aa4ea", "dependencies": [{ "name": "ring", "version": "0.16.20", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "untrusted", "version": "0.7.1", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null } }, { "advisory": { "id": "RUSTSEC-2023-0052", "package": "webpki", "title": "webpki: CPU denial of service in certificate path building", "description": "When this crate is given a pathological certificate chain to validate, it will\nspend CPU time exponential with the number of candidate certificates at each\nstep of path building.\n\nBoth TLS clients and TLS servers that accept client certificate are affected.\n\nThis was previously reported in\nhttps://github.com/briansmith/webpki/issues/69 and re-reported recently\nby Luke Malinowski.\n\nwebpki 0.22.1 included a partial fix and webpki 0.22.2 added further fixes.", "date": "2023-08-22", "aliases": ["GHSA-8qv2-5vq6-g2g7"], "related": ["CVE-2018-16875"], "collection": "crates", "categories": ["denial-of-service"], "keywords": ["certificate", "path building", "x509"], "cvss": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H", "informational": null, "references": [], "source": null, "url": null, "withdrawn": null, "license": "CC0-1.0" }, "versions": { "patched": [">=0.22.2"], "unaffected": [] }, "affected": null, "package": { "name": "webpki", "version": "0.22.0", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "f095d78192e208183081cc07bc5515ef55216397af48b873e5edcd72637fa1bd", "dependencies": [{ "name": "ring", "version": "0.16.20", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "untrusted", "version": "0.7.1", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null } }] }, "warnings": { "unmaintained": [{ "kind": "unmaintained", "package": { "name": "aes-soft", "version": "0.6.4", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "be14c7498ea50828a38d0e24a765ed2effe92a705885b57d029cd67d45744072", "dependencies": [{ "name": "cipher", "version": "0.2.5", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "opaque-debug", "version": "0.3.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": { "id": "RUSTSEC-2021-0060", "package": "aes-soft", "title": "`aes-soft` has been merged into the `aes` crate", "description": "Please use the `aes` crate going forward. The new repository location

is at:\n\nhttps://github.com/RustCrypto/block-ciphers/tree/master/aes\n\nAES-NI is now autodetected at runtime on `i686` / `x86-64` platforms.\nIf AES-NI is not present, the `aes` crate will fallback to a constant-time\nportable software implementation.\n\nTo force the use of a constant-time portable implementation on these platforms,\neven if AES-NI is available, use the new `force-soft` feature of the `aes` \ncrate to disable autodetection.", "date": "2021-04-29", "aliases": [], "related": [], "collection": "crates", "categories": [], "keywords": [], "cvss": null, "informational": "unmaintained", "references": [], "source": null, "url": "https://github.com/RustCrypto/block-ciphers/pull/200", "withdrawn": null, "license": "CC0-1.0" }, "affected": null, "versions": { "patched": [], "unaffected": [] } }, { "kind": "unmaintained", "package": { "name": "aesni", "version": "0.10.0", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "ea2e11f5e94c2f7d386164cc2aa1f97823fed6f259e486940a71c174dd01b0ce", "dependencies": [{ "name": "cipher", "version": "0.2.5", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "opaque-debug", "version": "0.3.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": { "id": "RUSTSEC-2021-0059", "package": "aesni", "title": "`aesni` has been merged into the `aes` crate", "description": "Please use the `aes` crate going forward. The new repository location is at:\n\nhttps://github.com/RustCrypto/block-ciphers/tree/master/aes\n\nAES-NI is now autodetected at runtime on `i686` / `x86-64` platforms.\nIf AES-NI is not present, the `aes` crate will fallback to a constant-time\nportable software implementation.\n\nTo prevent this fallback (and have absence of AES-NI result in an illegal\ninstruction crash instead), continue to pass the same RUSTFLAGS which were\npreviously required for the `aesni` crate to compile:\n\n`\nRUSTFLAGS=-Ctarget-feature=+aes,+ssse3\n`", "date": "2021-04-29", "aliases": [], "related": [], "collection": "crates", "categories": [], "keywords": [], "cvss": null, "informational": "unmaintained", "references": [], "source": null, "url": "https://github.com/RustCrypto/block-ciphers/pull/200", "withdrawn": null, "license": "CC0-1.0" }, "affected": null, "versions": { "patched": [], "unaffected": [] } }, { "kind": "unmaintained", "package": { "name": "ansi_term", "version": "0.12.1", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "d52a9bb7ec0cf484c551830a7ce27bd20d67eac647e1befb56b0be4ee39a55d2", "dependencies": [{ "name": "winapi", "version": "0.3.9", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": { "id": "RUSTSEC-2021-0139", "package": "ansi_term", "title": "ansi_term is Unmaintained", "description": "The maintainer has advised that this crate is deprecated and will not receive any maintenance.\n\nThe crate does not seem to have much dependencies and may or may not be ok to use as-is.\n\nLast release seems to have been three years ago.\n\n## Possible Alternative(s)\n\n The below list has not been vetted in any way and may or may not contain alternatives;\n\n - ansiterm\n - anstyle\n - console\n - nu-ansi-term\n - owo-colors\n - stylish\n - yansi\n\n## Dependency Specific Migration(s)\n\n - structopt, clap2", "date": "2021-08-18", "aliases": [], "related": [], "collection": "crates", "categories": [], "keywords": [], "cvss": null, "informational": "unmaintained", "references": [], "source": null, "url": "https://github.com/ogham/rust-ansi-term/issues/72", "withdrawn": null, "license": "CC0-1.0" }, "affected": null, "versions": { "patched": [], "unaffected": [] } }, { "kind": "unmaintained", "package": { "name": "mach", "version": "0.3.2", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "b823e83b2affd8f40a9ee8c29dbc56404c1e34cd2710921f2801e2cf29527afa", "dependencies": [{ "name": "libc", "version": "0.2.147", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": { "id": "RUSTSEC-2020-0168", "package": "mach", "title": "mach is unmaintained", "description": "Last release was almost 4 years ago.\n\nMaintainer(s) seem to be completely unreachable. \n\n## Possible Alternative(s)\n\nThese may or may not be suitable alternatives and have not been vetted in any way;\n- mach2 - direct fork", "date": "2020-07-14", "aliases": [], "related": [], "collection": "crates", "categories": [], "keywords": [], "cvss": null, "informational": "unmaintained", "references": [], "source": null, "url": "https://github.com/fitzgen/mach/issues/63", "withdrawn": null, "license": "CC0-1.0" }, "affected": null, "versions": { "patched": [], "unaffected": [] } }, { "kind": "unmaintained", "package": { "name": "parity-wasm", "version": "0.45.0", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "e1ad0aff30c1da14b1254fcb2af73e1fa9a28670e584a626f53a369d0e157304", "replace": null }, "advisory": { "id": "RUSTSEC-2022-0061", "package": "parity-wasm", "title": "Crate `parity-wasm` deprecated by the author", "description": "This PR explicitly deprecates `parity-wasm`.\nThe author recommends switching to wasm-tools.", "date": "2022-10-01", "aliases": [], "related": [], "collection": "crates", "categories": [], "keywords": [], "cvss": null, "informational": "unmaintained", "references": [], "source": null, "url": "https://github.com/paritytech/parity-wasm/pull/334", "withdrawn": null, "license": "CC0-1.0" }, "affected": null, "versions": { "patched": [], "unaffected": [] } }], "unsound": [{ "kind": "unsound", "package": { "name": "atty", "version": "0.2.14", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "d9b39be18770d11421cdb1b9947a45dd3f37e93092cbf377614828a319d5fee8", "dependencies": [{ "name": "hermit-abi", "version": "0.1.19", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "libc", "version": "0.2.147", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "winapi", "version": "0.3.9", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": { "id": "RUSTSEC-2021-0145", "package": "atty", "title": "Potential unaligned read", "description": "On windows, `atty` dereferences a potentially unaligned pointer.\n\nIn practice however, the pointer won't be unaligned unless a custom global allocator is used.\n\nIn particular, the `System` allocator on windows uses `HeapAlloc`, which guarantees a large enough alignment.\n\n# atty is Unmaintained\n\nA Pull Request with a fix has been provided over a year ago but the maintainer seems to be unreachable.\n\nLast release of `atty` was almost 3 years ago.\n\n## Possible Alternative(s)\n\nThe below list has not been vetted in any way and may or may not contain alternatives;\n\n - std::io::IsTerminal - Stable since Rust 1.70.0\n - is-terminal - Standalone crate supporting Rust older than 1.70.0", "date": "2021-07-04", "aliases": ["GHSA-g98v-hv3f-hcfr"], "related": [], "collection": "crates", "categories": [], "keywords": ["unaligned-read"], "cvss": null, "informational": "unsound", "references": ["https://github.com/softprops/atty/pull/51", "https://github.com/softprops/atty/issues/57"], "source": null, "url": "https://github.com/softprops/atty/issues/50", "withdrawn": null, "license": "CC0-1.0" }, "affected": { "arch": [], "os": ["windows"], "functions": {} }, "versions": { "patched": [], "unaffected": [] } }], "yanked": [{ "kind": "yanked", "package": { "name": "ahash", "version": "0.7.6", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "fcb51a0695d8f838b1ee009b3fbf66bda078cd64590202a864a8f3e8c4315c47", "dependencies": [{ "name": "getrandom", "version": "0.2.10", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "once_cell", "version": "1.18.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "version_check", "version": "0.9.4", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": null, "affected": null, "versions": null }, { "kind": "yanked", "package": { "name": "ahash", "version": "0.8.3", "source": "registry+https://github.com/rust-lang/crates.io-index", "checksum": "2c99f64d1e06488f620f932677e24bc6e2897582980441ae90a671415bd7ec2f", "dependencies": [{ "name": "cfg-if", "version": "1.0.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "getrandom", "version": "0.2.10", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "once_cell", "version": "1.18.0", "source": "registry+https://github.com/rust-lang/crates.io-index" }, { "name": "version_check", "version": "0.9.4", "source": "registry+https://github.com/rust-lang/crates.io-index" }], "replace": null }, "advisory": null, "affected": null, "versions": null }] } }

Test Suite Results

Run `cargo test -p pallet-unified-accounts`

```
    Finished test [unoptimized + debuginfo] target(s) in 3.96s
     Running unittests src/lib.rs (target/debug/deps/pallet_unified_accounts-d84952471474274f)

running 10 tests
test mock::__construct_runtime_integrity_test::runtime_integrity_tests ... ok
test tests::account_default_claim_should_not_work_if_collision ... ok
test tests::eip712_signature_verify_works ... ok
test tests::account_default_claim_works ... ok
test tests::on_killed_account_hook ... ok
test tests::account_claim_should_work ... ok
test tests::frontrun_attack_should_not_be_possible ... ok
test tests::connecting_mapped_accounts_should_not_work ... ok
test tests::static_lookup_works ... ok
test tests::replay_attack_should_not_be_possible ... ok

test result: ok. 10 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.05s

   Doc-tests pallet-unified-accounts

running 0 tests

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

Changelog

- 2023-11-27 - Initial report
- 2023-12-12 - Fix review update

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp