Quantstamp

Sequence - WAAS

Executive Summary

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

Type	Wallet
Timeline	2024-07-15 through 2024-08-06
Language	Go
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Quantstamp Sequence Audit June 2024 WaaS non-OIDC provider support New WaaS authentication flows Docs
Source Code	• 0xsequence/waas-authenticator #69cf96e
Auditors	• Nikita Belenkov Auditing Engineer • Valerian Callens Senior Auditing Engineer • Andy Lin Senior Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	7 Fixed: 5 Acknowledged: 2
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	4 Fixed: 3 Acknowledged: 1
Undetermined severity findings	1 Fixed: 1
Informational findings	1 Acknowledged: 1

Summary of Findings

We reviewed the Sequence Embedded Wallet (WaaS) implementation code, which handles authentication and authorization for the user wallet and holds one of the private keys for each of the projects (tenant). Users can send "intents," and the service will act upon these intents, such as signing and sending transactions. The service is deployed and run with the AWS Nitro enclave, which helps secure its memory and CPU access. The AWS Nitro implementation is used in conjunction with a Trusted Third Party KMS configuration, which Quantstamp has also previously audited. This design mitigates some of the inherent weaknesses of using AWS Nitro on its own.

One of the most significant risks to such a system is the centralization concern of a non-custodial wallet. If the Sequence systems go down, the wallet could potentially be unavailable to the user. To combat this, the Sequence team has added a few protections, such as a feature that enables either party (Sequence or the game partner) to take signing control of the wallets through a time-lock contract. Our team has previously audited this time lock contract. There is also a complete self-recovery tool in development.

We found the code well-written and well-modularized, making it easy to follow. Although there are tests, they do not fully cover all features. We suggest that the team provides higher coverage, including integration tests with other services. During the audit, we identified some issues, and we recommend the team address all of them.

Fix Review Update

The team has either fixed or acknowledged all the issues that have been highlighted in this report

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-1	**Playfab Authentication Can Be Bypassed**	Medium	Fixed
SEQ-2	**Denial of Service Risk Due to Lack of Http Client Timeout**	Low	Fixed
SEQ-3	**Unclear Data when Calling `UpdateTenant()`**	Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-4	**Incorrect Order of Terms in Subtraction Leads to Incorrect Calculation of `IntentResponseAuthInitiated.ExpiresIn` in `InitiateAuth()` for Emails**	Low	Fixed
SEQ-5	**Intent Signature Malleability Is Possible**	Low	Acknowledged
SEQ-6	**Unhandled Errors**	Informational	Acknowledged
SEQ-7	**Values of Interest Returned via Errors and `Print()` Functions**	Undetermined	Fixed

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):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

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

Files Included

Repo: https://github.com/0xsequence/waas-authenticator/(69cf96ee0f24d7a8f9f87fa62a86b9222df00844) Files: rpc/*

Operational Considerations

1. There should be end-to-end TLS encryption; otherwise, some authentication secrets may be revealed. Additionally, the call to the WAAS service needs protection to ensure the response is not corrupted.
2. AWS should be trusted, as the entire architecture relies on AWS infrastructure to encrypt/decrypt and handle authorizations.
3. The WAAS API is designed to be idempotent so that the client can retry with Auth service's API. We also assume that the client will attempt to retry on expected failures.
4. When deploying in production the system, `s.Config.Service.DebugProfiler` should be set to false to avoid exposing the endpoint `/debug`.
5. Access to the wallet is reliant on the service to be up and running. So if the Sequence Wallet goes down, the users cannot access their wallet or their funds.
6. It is potentially possible for the email service of Sequence to get blacklisted, as an attacker could generate thousands of fake emails that would cause the email anti-spamming system to blacklist the source.

Key Actors And Their Capabilities

Within the service's overall architecture, there are a few services that this Auth service interacts with:

- **Builder Service**: This is the source that can call the "admin" RPC endpoints. For instance, whenever a project is registered, it will need to go through the builder service to create a tenant in the auth service.
- **WAAS Service**: This is the main downstream service of this Auth service. This will help the auth service to create the "project wallet" and also handle the orchestration tasks with other services like the Guard Service.
- **Guard Service**: There is no direct interaction from the Auth service, but WAAS service will call Guard service as one of the its downstream services. The Guard will also need to sign the transaction to fulfil the 2/2 multi-signature on-chain wallet.

There are different kinds of "users" defined within this auth service:

- **Tenants**: Each project would have one tenant. This is the highest in the hierarchy of the user/account systems. A tenant will have a "parent wallet" in the auth service with the private key encrypted by KMS.
- **User**: Users belongs to a tenant.
- **Wallet**: Each user will have one wallet. The wallet address is derived from the user ID.
- **Account/Identity**: Each authentication method will require one account and identity ID for each user.
- **Session**: Once authenticated, a user can open a session until it expires. The session ID will be the user's public key data on its client side. Within the session time, the user only needs to sign the intent and does not need to authenticate again. A user can open several sessions on different devices with different devices' private keys.

Findings

SEQ-1 Playfab Authentication Can Be Bypassed

Medium Fixed

Update
The team fixed the issue as recommended in the commits `65cfb43` and `c016245`.

File(s) affected: `rpc/auth/playfab/playfab_api.go`, `rpc/auth/playfab/provider.go`, `rpc/awscreds/provider.go`

Description: In `auth/playfab/provider.go`, the `Verify()` function ensures the `answer` (which is the Playfab session ticket) is valid by calling their API in the `playfab_api.go:getAccountInfo()` function. However, in the `getAccountInfo()` function, it does not check the response's HTTP status. A non-2xx status code doesn't cause an error in the `client.Do()` call (see: doc). Also, the line `json.NewDecoder(res.Body).Decode(&resp)` will not error as long as the response does not have the same JSON key. The `resp` will still have default values after the JSON decoding since no data is being written. Since the `getAccountInfo()` function does not return an error, the `Verify()` function will be considered valid and return an `identity` with `Subject` and `Email` fields being empty values.

The same pattern that misses the check of response status can also be found in the `getAWSCredential()` and `getInstanceProfileName()` functions of `awscreds/provider.go`. It appears to have less of an impact there as the `Retrieve()` function, which calls the `getAWSCredential()` function, will return empty `aws.Credentials{}` on error, so the result will be the same if the call failed.

Recommendation: Check the return status to be 2xx and return an error if not before further processing in the `getAccountInfo()`, `getAWSCredential()`, and `getInstanceProfileName()` functions.

SEQ-2 Denial of Service Risk Due to Lack of Http Client Timeout

Low Fixed

Update
The team fixed the issue as recommended in the commit `66a7447`.

File(s) affected: `rpc/rpc.go`, `cmd/waas-auth/main.go`

Description: The `rpc.go:New()` function uses the `http.DefaultClient` if the `client` input is not provided (`nil`). However, the default client has its `Timeout` field set to zero, which means "no timeout". Although the current code setup calls `rpc.go:New()` in `waas-`

`auth/main.go` with a customized HTTP client, that client also does not set the `Timeout` field.

This could pose a denial of service risk if the downstream service has issues and hangs on responding, causing this auth service to hang or, even worse, run out of resources as all are waiting. For more details, refer to this blog post.

Recommendation: Add a practical `Timeout` setting to both `main.go` and `rpc.go`.

SEQ-3 Unclear Data when Calling `UpdateTenant()`

Low Fixed

> **Update**
> The team set the "CreatedAt" field, which later will be used as "UpdatedAt" to `time.Now()` in the commit `654df99`.

File(s) affected: `rpc/admin.go`

Description: In `rpc/admin::UpdateTenant()`, the admin can update some attributes of a registered tenant. However: a. `tnt.CreatedAt()` is not updated to `time.Now()`, even if it is a new version of the `TenantData`; b. `retTenant.UpdatedAt` takes the value of `tnt.CreatedAt`; These two operations are confusing and should be clarified as working as expected.

Recommendation: Consider confirming if it is the expected behavior, or adapt the code.

SEQ-4 Incorrect Order of Terms in Subtraction Leads to Incorrect Calculation of `IntentResponseAuthInitiated.ExpiresIn` in `InitiateAuth()` for Emails

Low Fixed

> **Update**
> The team fixed the issue as recommended in the commit `2466371`.

File(s) affected: `rpc/auth/email/provider.go`

Description: In `rpc/auth/email/provider::InitiateAuth()`, the value of `verifCtx.ExpiresAt` is set to `time.Now().Add(30 * time.Minute)`. However, a few lines later, `res.ExpiresIn` is set to `int(time.Now().Sub(verifCtx.ExpiresAt).Seconds())`. This operation is incorrect since `time.Now() < verifCtx.ExpiresAt`.

Recommendation: Consider inverting the terms.

SEQ-5 Intent Signature Malleability Is Possible

Low Acknowledged

> **Update**
> The team acknowledged the issue with the following statement:
>
> ```
> We consider this a non-issue since signatures aren't used for replay protection, the system does not
> assume non-malleable signatures
> ```

File(s) affected: `rpc/intents.go`, `rpc/utils.go`

Description: The `ecdsa.Verify()` function does not protect against signature malleability attacks by itself when recovering either `p256k1` or `p256r1` signatures. ECDSA signatures are malleable, meaning they can be modified without invalidating the signature. For every valid signature (`r`, `s`), there exists another valid signature with a different `s` value. Ethereum standards recommend normalizing the `s` value to its lower half, ensuring that each signature uniquely corresponds to the signer's address, thus reducing malleability risks.

The `intent.Signers()` function does not include validation of the `s` value to prevent signature malleability, therefore it is potentially possible to replay an intent signature. If the nonce accounting is correctly enforced, this does not lead to unexpected behaviour.

Recommendation: Normalize the `s` value to its lower half.

SEQ-6 Unhandled Errors

Informational Acknowledged

> **Update**
> The team is currently working on fixing this issue.

Alert

It is important to note that the logic control flow will differ as the current implementation ignores the error and continues executing. That might lead to unexpected behaviors and should be handled with care and test cases.

File(s) affected: `rpc/identity_provider.go`, `rpc/admin.go`

Description: It is a good practice for calls to return an error type that should be checked to make sure that the call had the expected behavior. There are certain calls that do not check the returned errors:

1. `w.Write()` in `emptyHandler()` and `indexHandler()`
2. `json.NewEncoder(w).Encode()` in `handleOpenidConfiguration()` and `handleJWKS()`

Recommendation: Consider handling the errors returned from these functions.

SEQ-7
Values of Interest Returned via Errors and `Print()` Functions

Undetermined | Fixed

Update

Most points listed out were fixed in the commit `57c0b72`.

File(s) affected: `rpc/signing/kms.go`, `rpc/auth/oidc.go`, `rpc/accounts.go`

Description: In multiple locations, values of interest can be returned to users and may provide hints to malicious users about the system. For instance:

1. In `rpc/signing/kms::PublicKey()`: `return nil, fmt.Errorf("invalid public key type: %T", jwtKey)`
2. In `rpc/auth/oidc::withSessionHash()`: `return jwt.NewValidationError(fmt.Errorf("nonce not satisfied: %s != %s", nonceVal, expectedSessionHash))`
3. In `rpc/accounts::deleteAccountSessions()`:
   - `fmt.Printf("sessions of user %s: %+v\n", userID, sessions)`
   - `fmt.Println("deleteAccountSessions: skipping session", sess.Identity)`

Recommendation: Consider if these pieces of information should or should not be displayed or returned to the user.

Auditor Suggestions

S1 General Suggestions

Fixed

Update

The team followed all of the suggestions in the commit `a637d8f`.

File(s) affected: `rpc/crypto/crypto.go`, `rpc/oidc/keyset.go`, `rpc/oidc/provider.go`, `rpc/oidc/legacy.go`, `rpc/rpc.go`

Description: Here are some best practices that we suggest following:

1. In `crypto/crypto.go:EncryptData()`, the error message `fmt.Errorf("AES decrypt: %w", err)` should probably be "encrypt" instead of "decrypt". Consider fixing the message.
2. In `oidc/keyset.go`, the `operationKeySet` struct has the `cachedSet` field which seems never to be read. It is only written in the `getCachedSet()` function but never used later. Alternatively, please revisit if `getCachedSet()` should attempt to read from the `cachedSet` first.
3. In `oidc/provider.go` and `oidc/legacy.go`, the `ValidateTenant()` function, instead of doing `ctx, cancel := context.WithCancel(ctx)`, might benefit from using `wg, ctx := errgroup.WithContext(ctx)` to ensure there is a context cancellation from the `errgroup` so that it can stop other tasks if any of them fails (see this blog post). Note that the `cancel` in the current implementation is not used.
4. The `rpc.go:Ping()` function seems unused. There is a `status.go:healthHandler()` function that appears to replace this `Ping()` function. Consider removing the unused function.
5. In the `rpc.go:newOtelTracerProvider()` function, the `ctx` input is unused. Consider removing it from the function input.

Recommendation: Follow the best practices as mentioned in the description section.

S2 The Logging Level of the Rpc Is Debuglevel by Default

Acknowledged

Update

The team has commented the following:

```
As the service runs within a secure nitro enclave, no logs are output in production at any level
```

Description: By default, the logging level of an RPC when it is created in `rpc/rpc::New()` is hardcoded to `zerolog.LevelDebugValue`. Since using an improper logging level could lead to using too much memory space if it is too verbose and detecting too few events if it is insufficiently verbose, consider setting this value based on a dynamic input.

Recommendation: Consider setting this value based on a dynamic input.

S3 Production Readiness of the Code

Acknowledged

Update
The team has acknowledged the issue

File(s) affected: `rpc/intents.go`, `rpc/admin.go`, `rpc/accounts.go`, `rpc/rpc.go`

Description: There are multiple places where TODOs are still left in the codebase. These should be addressed before deployment:

1. `RPC.deleteAccountSessions()`
2. `admin.CreateTenant()`
3. `admin.UpdateTenant()`
4. `intents.SendIntent()`

The codebase also contains functionality that is intended for testing purposes, which should be removed before launch, such as this AWS test credentials:

```
awsconfig.WithCredentialsProvider(&awscreds.StaticProvider{
            AccessKeyID:     "test",
            SecretAccessKey: "test",
            SessionToken:    "test",
```

Recommendation: Fix or remove `TODO` statements and test functionality.

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).

Toolset

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

Setup

Tool Setup:
- GoSec 2.4.0

Steps taken to run the tools:
- Install gosec: go get github.com/securego/gosec/cmd/gosec

- run gosec againt all modules: gosec ./...

Automated Analysis

GoSec

5 issues have been identified and, and, when relevant, have been included in the report.

Test Suite Results

The test suite contains 64 tests, all of which were successfully passed.

```
GOGC=off go clean -testcache
GOGC=off go test -v -run= ./...
?       github.com/0xsequence/waas-authenticator    [no test files]
?       github.com/0xsequence/waas-authenticator/cmd/builder-mock   [no test files]
?       github.com/0xsequence/waas-authenticator/cmd/jwt-util   [no test files]
?       github.com/0xsequence/waas-authenticator/cmd/waas-auth  [no test files]
?       github.com/0xsequence/waas-authenticator/config [no test files]
?       github.com/0xsequence/waas-authenticator/proto  [no test files]
?       github.com/0xsequence/waas-authenticator/proto/builder  [no test files]
?       github.com/0xsequence/waas-authenticator/proto/clients  [no test files]
?       github.com/0xsequence/waas-authenticator/proto/waas [no test files]
?       github.com/0xsequence/waas-authenticator/data   [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/access [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/attestation    [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/auth   [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/auth/guest [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/auth/oidc  [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/auth/playfab   [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/crypto [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/migration  [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/tenant [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/tracing    [no test files]
?       github.com/0xsequence/waas-authenticator/rpc/waasapi    [no test files]
=== RUN   TestRPC_GetTenant
=== RUN   TestRPC_GetTenant/ExistingTenant
--- PASS: TestRPC_GetTenant (2.74s)
    --- PASS: TestRPC_GetTenant/ExistingTenant (0.07s)
    --- PASS: TestRPC_GetTenant/ExistingTenantWithAuthConfig (0.02s)
    --- PASS: TestRPC_GetTenant/MissingTenant (0.01s)
=== RUN   TestRPC_CreateTenant
=== RUN   TestRPC_CreateTenant/TenantAlreadyExists
=== RUN   TestRPC_CreateTenant/InvalidProvider
=== RUN   TestRPC_CreateTenant/InvalidOrigin
=== RUN   TestRPC_CreateTenant/InvalidPassword
=== RUN   TestRPC_CreateTenant/Success
=== RUN   TestRPC_CreateTenant/SuccessWithPassword
--- PASS: TestRPC_CreateTenant (1.00s)
    --- PASS: TestRPC_CreateTenant/TenantAlreadyExists (0.01s)
    --- PASS: TestRPC_CreateTenant/InvalidProvider (0.02s)
    --- PASS: TestRPC_CreateTenant/InvalidOrigin (0.01s)
    --- PASS: TestRPC_CreateTenant/InvalidPassword (0.01s)
    --- PASS: TestRPC_CreateTenant/Success (0.29s)
    --- PASS: TestRPC_CreateTenant/SuccessWithPassword (0.40s)
=== RUN   TestEmailAuth
=== RUN   TestEmailAuth/Success
=== RUN   TestEmailAuth/CaseInsensitive
=== RUN   TestEmailAuth/IncorrectCode
=== RUN   TestEmailAuth/MultipleAttempts
=== RUN   TestEmailAuth/TooManyAttempts
--- PASS: TestEmailAuth (12.62s)
    --- PASS: TestEmailAuth/Success (0.78s)
    --- PASS: TestEmailAuth/CaseInsensitive (1.42s)
    --- PASS: TestEmailAuth/IncorrectCode (4.44s)
    --- PASS: TestEmailAuth/MultipleAttempts (5.43s)
    --- PASS: TestEmailAuth/TooManyAttempts (0.55s)
```

```
=== RUN   TestGuestAuth
=== RUN   TestGuestAuth/Success
--- PASS: TestGuestAuth (1.04s)
    --- PASS: TestGuestAuth/Success (1.04s)
=== RUN   TestOIDCAuth
=== RUN   TestOIDCAuth/Success
--- PASS: TestOIDCAuth (0.33s)
    --- PASS: TestOIDCAuth/Success (0.33s)
=== RUN   TestStytchAuth
=== RUN   TestStytchAuth/Success
--- PASS: TestStytchAuth (0.44s)
    --- PASS: TestStytchAuth/Success (0.44s)
=== RUN   TestPlayFabAuth
=== RUN   TestPlayFabAuth/Success
=== RUN   TestPlayFabAuth/PlayFabReturns500
=== RUN   TestPlayFabAuth/PlayFabReturnsEmptyID
--- PASS: TestPlayFabAuth (0.66s)
    --- PASS: TestPlayFabAuth/Success (0.22s)
    --- PASS: TestPlayFabAuth/PlayFabReturns500 (0.28s)
    --- PASS: TestPlayFabAuth/PlayFabReturnsEmptyID (0.16s)
=== RUN   TestRPC_SendIntent_GetIdToken
--- PASS: TestRPC_SendIntent_GetIdToken (0.34s)
=== RUN   TestMigrationOIDCToStytch
=== RUN   TestMigrationOIDCToStytch/WithoutConfig
=== RUN   TestMigrationOIDCToStytch/WithoutConfig/NoContinuousMigration
=== RUN   TestMigrationOIDCToStytch/ContinuousMigration
=== RUN   TestMigrationOIDCToStytch/OneTimeMigration
--- PASS: TestMigrationOIDCToStytch (1.52s)
    --- PASS: TestMigrationOIDCToStytch/WithoutConfig (0.28s)
        --- PASS: TestMigrationOIDCToStytch/WithoutConfig/NoContinuousMigration (0.28s)
    --- PASS: TestMigrationOIDCToStytch/ContinuousMigration (0.36s)
    --- PASS: TestMigrationOIDCToStytch/OneTimeMigration (0.88s)
=== RUN   TestMigrationEmail
=== RUN   TestMigrationEmail/ContinuousMigration
=== RUN   TestMigrationEmail/OneTimeMigration
--- PASS: TestMigrationEmail (1.21s)
    --- PASS: TestMigrationEmail/ContinuousMigration (0.46s)
    --- PASS: TestMigrationEmail/OneTimeMigration (0.76s)
=== RUN   TestRPC_SendIntent_SendTransaction
--- PASS: TestRPC_SendIntent_SendTransaction (0.48s)
=== RUN   TestLegacyAuth
=== RUN   TestLegacyAuth/EmailAlreadyInUse
=== RUN   TestLegacyAuth/WithValidNonce
=== RUN   TestLegacyAuth/WithInvalidNonce
=== RUN   TestLegacyAuth/WithMissingNonce
=== RUN   TestLegacyAuth/WithVerifiedEmail
=== RUN   TestLegacyAuth/MissingSignature
=== RUN   TestLegacyAuth/IssuerMissingScheme
=== RUN   TestLegacyAuth/EmailAlreadyInUseWithForceCreateAccount
=== RUN   TestLegacyAuth/Basic
=== RUN   TestLegacyAuth/WithInvalidIssuer
=== RUN   TestLegacyAuth/WithInvalidNonceButValidSessionAddressClaim
--- PASS: TestLegacyAuth (4.01s)
    --- PASS: TestLegacyAuth/EmailAlreadyInUse (0.43s)
    --- PASS: TestLegacyAuth/WithValidNonce (0.20s)
    --- PASS: TestLegacyAuth/WithInvalidNonce (0.64s)
    --- PASS: TestLegacyAuth/WithMissingNonce (0.52s)
    --- PASS: TestLegacyAuth/WithVerifiedEmail (0.32s)
    --- PASS: TestLegacyAuth/MissingSignature (0.27s)
    --- PASS: TestLegacyAuth/IssuerMissingScheme (0.37s)
    --- PASS: TestLegacyAuth/EmailAlreadyInUseWithForceCreateAccount (0.28s)
    --- PASS: TestLegacyAuth/Basic (0.38s)
    --- PASS: TestLegacyAuth/WithInvalidIssuer (0.22s)
    --- PASS: TestLegacyAuth/WithInvalidNonceButValidSessionAddressClaim (0.36s)
=== RUN   TestRPC_SendIntent_DropSession
=== RUN   TestRPC_SendIntent_DropSession/SameSession
=== RUN   TestRPC_SendIntent_DropSession/SameUser
=== RUN   TestRPC_SendIntent_DropSession/OtherUser
--- PASS: TestRPC_SendIntent_DropSession (1.66s)
    --- PASS: TestRPC_SendIntent_DropSession/SameSession (0.48s)
    --- PASS: TestRPC_SendIntent_DropSession/SameUser (0.45s)
    --- PASS: TestRPC_SendIntent_DropSession/OtherUser (0.73s)
```

```
=== RUN   TestRPC_SendIntent_ListSessions
--- PASS: TestRPC_SendIntent_ListSessions (0.68s)
=== RUN   TestRPC_SendIntent_SignMessage
--- PASS: TestRPC_SendIntent_SignMessage (0.28s)
PASS
ok      github.com/0xsequence/waas-authenticator/rpc    63.909s
=== RUN   TestExtractVerifier
=== RUN   TestExtractVerifier/user@example.com
=== RUN   TestExtractVerifier/user@example.com;0x1234
=== RUN   TestExtractVerifier/__USER@example.COM__;0x1234
--- PASS: TestExtractVerifier (0.00s)
    --- PASS: TestExtractVerifier/user@example.com (0.00s)
    --- PASS: TestExtractVerifier/user@example.com;0x1234 (0.00s)
    --- PASS: TestExtractVerifier/__USER@example.COM__;0x1234 (0.00s)
PASS
ok      github.com/0xsequence/waas-authenticator/rpc/auth/email 5.992s
=== RUN   TestProvider_Retrieve
--- PASS: TestProvider_Retrieve (0.00s)
PASS
ok      github.com/0xsequence/waas-authenticator/rpc/awscreds   0.587s
=== RUN   TestMiddleware
=== RUN   TestMiddleware/NoAcceptSignature
=== RUN   TestMiddleware/BasicAcceptSignature
=== RUN   TestMiddleware/FullAcceptSignature
=== RUN   TestMiddleware/ExtendedAcceptSignature
--- PASS: TestMiddleware (0.00s)
    --- PASS: TestMiddleware/NoAcceptSignature (0.00s)
    --- PASS: TestMiddleware/BasicAcceptSignature (0.00s)
    --- PASS: TestMiddleware/FullAcceptSignature (0.00s)
    --- PASS: TestMiddleware/ExtendedAcceptSignature (0.00s)
PASS
ok      github.com/0xsequence/waas-authenticator/rpc/signing    0.850s
```

Changelog

- 2024-08-09 - Initial report
- 2024-09-27 - Final report

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

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

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 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 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 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 and may not be represented as such. No third party is entitled to rely on the report in any any way, including for the purpose of making any decisions to buy or sell a product, 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 or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related 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 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.