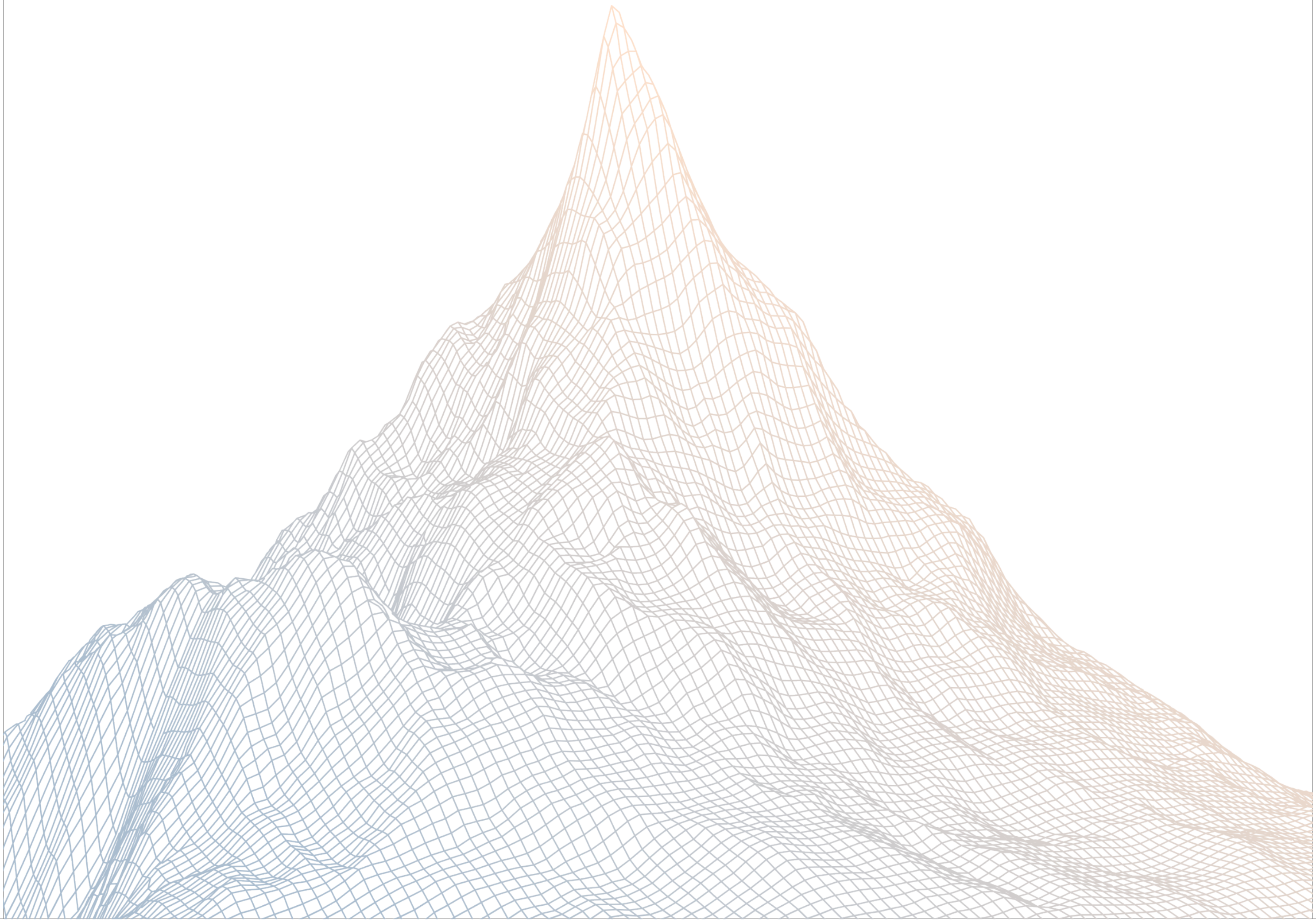
Zenith

Alien

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	September 15th to September 19th, 2025
AUDITED BY:	DadeKuma defsec kriko

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Alien

Alien is a highly scalable network of unique real humans designed to enable trust in the age of AI. The Alien Network implements CHVP, which grants every person in the world a unique Alien ID. The system is fair launched, decentralized, and programmable to support identity, payments, and trust across every sector of the internet.

2.2 Scope

The engagement involved a review of the following targets:

Target	solana-credential-signer
Repository	https://github.com/alien-id/solana-credential-signer
Commit Hash	`49a8a4e4d08e0da31467d92321ffeabc03759b4f`
Files	`programs/**/src/**.rs`

Target	solana-credential-signer-oracle
Repository	https://github.com/alien-id/solana-credential-signer-oracle
Commit Hash	`ccc913499b8ff179f42c15edd05476ee743320b8`

Files	`- internal/*`
	`- cmd/*`

2.3 Audit Timeline

September 15th, 2025	Audit start
September 19th, 2025	Audit end
September 29th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	13
Informational	4
Total Issues	**18**

3 Findings Summary

ID	Description	Status
M-1	Weak cryptographic key derivation from predictable secret	Resolved
L-1	Missing expiration mechanism for oracle signatures	Resolved
L-2	Restricted recipient in session registry close function	Acknowledged
L-3	Denial of service via unlimited signature requests	Acknowledged
L-4	GitHub access token leakage in Docker build	Acknowledged
L-5	Debug information in production builds	Resolved
L-6	Missing security headers	Resolved
L-7	Absence of TLS configuration	Acknowledged
L-8	Error information disclosure in request handlers	Resolved
L-9	Insecure default binding configuration	Acknowledged
L-10	Docker security misconfiguration	Acknowledged
L-11	Cross-origin resource sharing misconfiguration	Resolved
L-12	Expired and revoked sessions can generate valid signatures	Resolved
L-13	Signature replay attack vector	Resolved
I-1	Redundant instruction parameter deserialization in CreateAttestation	Resolved
I-2	Unnecessary account initialization for credential_signer PDA	Resolved
I-3	Unnecessary instruction data passed to close_attestation	Resolved
I-4	Inefficient vector capacity allocation negates optimization benefits	Resolved

4 Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Weak cryptographic key derivation from predictable secret

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- config.go#L55

Description:

The oracle derives its Ed25519 private key using only SHA256 hash of a secret string, without proper key derivation function (KDF). SHA256 is a fast hash allowing ~10 billion attempts/second on GPUs. No salt means identical secrets produce identical keys across deployments. The example configuration contains weak secret "test_secret_key". An attacker who obtains or brute-forces the secret can regenerate the same private key and forge any attestation signatures.

Recommendations:

Replace SHA256 with proper KDF like Argon2id or PBKDF2 with salt. Add domain separation. Implement HSM integration for key management. Validate secret strength at startup and reject known weak values.

Alien: Resolved with PR-6. KDF logic, domain separation, and salt have been implemented. The SECRET_KEY and KEY_SALT are specified uniquely with strictness and uniqueness checks and automatically in env variables in AWS deployment, with future use of HSM. HSM will be integrated in the future with proper infrastructure on our side.

Zenith: Verified.

4.2 Low Risk

A total of 13 low risk findings were identified.

[L-1] Missing expiration mechanism for oracle signatures

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/credential_signer/src/lib.rs

Description:

Oracle signatures currently lack an expiration mechanism, allowing users to reuse the same signature indefinitely for creating/closing attestations, and repeating these operations without limitation.

No practical attacks or immediate impacts have been identified. However, the absence of expiration creates a potential security risk by allowing signatures to remain valid permanently.

Recommendations:

Consider implementing an expiration timestamp for the oracle signature.

Alien: Resolved with PR-1 and PR-7.

Zenith: Verified.

[L-2] Restricted recipient in session registry close function

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- programs/credential_signer/src/lib.rs

Description:

The session registry allows funds to be sent to a custom recipient when a session is removed, which does not necessarily need to be the original signer.

However, the current implementation passes the payer address to the function, effectively restricting the recipient to only the initial creator of the session.

This limits the intended feature and prevents use cases where funds should be sent to a different account.

Recommendations:

Consider modifying the `close_attestation` function to support an optional custom recipient parameter.

Alien: Acknowledged. This is not a feature, so it is not required in the implementation.

[L-3] Denial of service via unlimited signature requests

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- /api/server.go

Description:

No rate limiting on /sign endpoint allows unlimited signature requests. Attackers can exhaust resources, perform timing attacks, or abuse the service to create massive numbers of attestations.

Recommendations:

Add application-level rate limiting using golang.org/x/time/rate. Implement per-IP and per-session-address limits. Add exponential backoff for failed attempts.

Alien: Resolved with PR-11.

Zenith: Verified.

[L-4] GitHub access token leakage in Docker build

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Dockerfile

Description:

GitHub access tokens passed as Docker build arguments and written to .netrc file in plaintext. These credentials remain in Docker image layers even in multi-stage builds, exposing GitHub access tokens that could allow unauthorized access to private repositories containing sensitive code.

Recommendations:

Replace build args with secure secret handling: RUN —mount=type=secret,id=github_token. Never write credentials to files in Docker layers.

Alien: Not possible. Our infrastructure is built on this and provides automatic protection.

Zenith: Acknowledged.

[L-5] Debug information in production builds

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Dockerfile

Description:

Go build flags don't strip debug information or symbols. Production binary contains debugging symbols that could help attackers reverse engineer the application or understand internal structure.

Recommendations:

Add build flags: -ldflags="-s -w" to strip symbols and debug info.

Alien: Resolved with PR-10.

Zenith: Verified.

[L-6] Missing security headers

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- server.go#L61

Description:

The server lacks mandatory security headers that enforce HTTPS usage and prevent downgrade attacks. Missing headers include:

- Strict-Transport-Security (HSTS)
  - Content-Security-Policy
  - X-Frame-Options
  - X-Content-Type-Options
  - Referrer-Policy

Recommendations:

Add comprehensive security headers middleware.

Alien: Implemented with PR-9. Strict-Transport-Security - not implemented. Our AWS infrastructure does it automatically.

Zenith: Verified.

[L-7] Absence of TLS configuration

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- server.go#L34

Description:

The HTTP server is configured without any TLS support, running exclusively on unencrypted HTTP. All sensitive data including session signatures, Solana addresses, and oracle signatures are transmitted in plaintext over the network.

Recommendations:

Implement TLS configuration.

Alien: Cannot be implemented. Our AWS infrastructure does this automatically.

Zenith: Acknowledged.

[L-8] Error information disclosure in request handlers

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- server.go#L57

Description:

Request and response error handlers expose internal error details directly to clients via http.Error(w, err.Error(), ...). This can leak sensitive information including file paths, internal state, database schemas, or dependency versions. Stack traces and Go runtime information may be included in error responses.

Recommendations:

Create error mapping that converts internal errors to safe external messages. Add error correlation IDs to link external responses with internal logs. Never expose `err.Error()` directly to clients.

Alien: Resolved with PR-10.

Zenith: Verified.

[L-9] Insecure default binding configuration

`SEVERITY:` Low	`IMPACT:` Low
`STATUS:` Acknowledged	`LIKELIHOOD:` Low

Target

- api/server.go#L36

Description:

Server defaults to binding on 0.0.0.0:8080 which exposes the service on all network interfaces including public interfaces. This makes the oracle accessible from any network the host is connected to, bypassing intended network segmentation.

Recommendations:

Change default host to 127.0.0.1. Add validation to reject wildcard binding unless explicitly configured. Implement network interface detection and warnings for public interface binding.

Alien: Cannot be implemented. Our infrastructure monitors network binding security, and the specified 0.0.0.0 is required for our infrastructure.

Zenith: Acknowledged.

[L-10] Docker security misconfiguration

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Dockerfile

Description:

Docker container may run as root user, exposing host system if container escape occurs. No security scanning or minimal base images specified. Missing security headers and container hardening configurations.

Recommendations:

Use minimal base images. Run as non-root user. Add security labels and resource limits. Implement read-only filesystem where possible. Add container security scanning to CI/CD pipeline.

Alien: Half of this is implemented on our AWS infrastructure side. Other half is will be implemented in future according to need.

Zenith: Acknowledged

[L-11] Cross-origin resource sharing misconfiguration

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- API endpoints (missing CORS configuration)

Description:

No CORS headers configured on API endpoints. Could enable browser-based attacks if service is accidentally exposed to web traffic. While oracle is likely not intended for browser access, missing CORS is a defense-in-depth failure.

Recommendations:

Implement restrictive CORS policy. Set Access-Control-Allow-Origin to specific domains only. Disable CORS entirely if browser access not needed. Add Content-Security-Policy headers.

Alien: Resolved with PR-9

Zenith: Verified.

[L-12] Expired and revoked sessions can generate valid signatures

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- authorization.go

Description:

Missing critical session validation checks. Code doesn't verify if session is active, expired, or revoked. No validation of session type or purpose. This allows use of inactive, expired, or wrong-purpose sessions to create attestations, potentially bypassing intended security controls.

Recommendations:

Add validation for session.Active status, expiry timestamp, revocation status, and session type/purpose. Implement comprehensive session state checks before signing.

Alien: Since protocol doesn't have expired/revoked sessions I added validation if session is active and created for solana provider (its purpose) on line authorization.go:16

Resolved with PR-8

Zenith: Verified.

[L-13] Signature replay attack vector

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- authorization.go

Description:

Oracle signatures lack replay protection mechanisms. The signed message is simple concatenation of sessionAddress + solanaAddress without timestamp.

Recommendations:

Include timestamp in the signed messages.

Alien: Resolved with PR-7 and PR-1

Zenith: Verified.

4.3 Informational

A total of 4 informational findings were identified.

[I-1] Redundant instruction parameter deserialization in CreateAttestation

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/credential_signer/src/lib.rs

Description:

The `CreateAttestation` accounts struct unnecessarily deserializes instruction parameters that are not used in any account constraints.

The `CreateAttestation` struct includes an `#[instruction(...)]` attribute that deserializes three parameters:

```
#[derive(Accounts)]
#[instruction(session_address: String, oracle_signature: [u8; 64], expiry:
    i64)]
pub struct CreateAttestation<'info> {
    // Account definitions...
}
```

However, these parameters are only used within the instruction handler function (`create_attestation`), not in the accounts struct itself. Anchor performs "eager deserialization" of all instruction parameters upfront, creating unnecessary computational overhead.

Recommendations:

Consider removing the `#[instruction(...)]` attribute from the `CreateAttestation` struct:

```
#[derive(Accounts)]
#[instruction(session_address:
String, oracle_signature: [u8; 64], expiry: i64)]
pub struct CreateAttestation<'info> {
    // Account definitions remain unchanged
}
```

The instruction parameters will still be available in the instruction handler function through the normal parameter list. This eliminates unnecessary deserialization overhead while maintaining identical functionality.

Alien: Resolved with PR-5

Zenith: Verified.

[I-2] Unnecessary account initialization for credential_signer PDA

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/credential_signer/src/lib.rs

Description:

The `credential_signer` account is unnecessarily initialized with allocated storage space despite never being written to or requiring ownership by the calling program, resulting in wasted rent costs.

In the `Initialize` struct, the `credential_signer` account is initialized with 40 bytes of storage:

```
#[account(
    init,
    payer = admin,
    space = 8 + 32,
    seeds = [b"credential_signer"],
    bump
)]
/// CHECK: PDA derived with seeds ["credential_signer"] used for CPI signing
pub credential_signer: AccountInfo<'info>,
```

However, analysis of the Solana Attestation Service program shows that the `authorized_signer` parameter (which receives this account) is only used for authentication purposes and never requires account ownership or data storage.

In `process_create_attestation`, the authorized signer is only validated through:

```
// From solana-attestation-service/processor/create_attestation.rs
verify_signer(authorized_signer, false)?;
credential.validate_authorized_signer(authorized_signer.key())?;
```

Similarly, in `process_close_attestation`:

```
// From solana-attestation-service/processor/close_attestation.rs
verify_signer(authorized_signer, false)?;
credential.validate_authorized_signer(authorized_signer.key())?;
```

This unnecessary initialization wastes `0.00116928` SOL in rent costs.

Recommendations:

Consider removing the `init`, `payer`, and `space` constraints from the `credential_signer` account since it only needs to exist as a PDA for signing purposes:

```
#[account(
   init,
   payer = admin,
   space = 8 + 32,
    seeds = [b"credential_signer"],
    bump
)]
/// CHECK: PDA derived with seeds ["credential_signer"] used for CPI signing
pub credential_signer: AccountInfo<'info>,
```

This eliminates the unnecessary account initialization and storage allocation while maintaining identical functionality for CPI signing operations.

Alien: Resolved with PR-4

Zenith: Verified.

[I-3] Unnecessary instruction data passed to close_attestation

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/credential_signer/src/lib.rs

Description:

The `close_attestation` instruction constructs instruction data with unnecessary bytes that are not processed by the target Solana Attestation Service program, resulting in wasted computation and data transmission.

In the `close_attestation` instruction, the code constructs instruction data containing both a discriminator and a 32-byte nonce:

```rust
let close_attestation_discriminator: u8 = 7;
let nonce_bytes = nonce.to_bytes();

let mut close_instruction_data = Vec::new();
close_instruction_data.push(close_attestation_discriminator);
close_instruction_data.extend_from_slice(&nonce_bytes);
```

This results in instruction data of 33 bytes total (1 byte discriminator + 32 bytes nonce).

However, examining the Solana Attestation Service program's entrypoint shows that discriminator 7 maps to the `process_close_attestation` function, which is called with `None` as the instruction data parameter:

```rust
// From solana-attestation-service/program/src/entrypoint.rs
7 ⇒ process_close_attestation(program_id, accounts, None)
```

The function does not deserialize or process any instruction data beyond the discriminator byte.

Recommendations:

Consider removing the unnecessary nonce bytes from the instruction data construction:

```
  let close_attestation_discriminator: u8 = 7;
let nonce_bytes = nonce.to_bytes();

  let mut close_instruction_data = Vec::new();
  close_instruction_data.push(close_attestation_discriminator);
close_instruction_data.extend_from_slice(&nonce_bytes);
```

This reduces the instruction data from 33 bytes to 1 byte, eliminating unnecessary data transmission and processing overhead while maintaining identical functionality.

Alien: Resolved with PR-3

Zenith: Verified.

[I-4] Inefficient vector capacity allocation negates optimization benefits

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/credential_signer/src/lib.rs
- programs/credential_signer/src/lib.rs

Description:

The `Vec::with_capacity` optimization in the CPI instruction data construction is rendered ineffective due to incorrect capacity calculation that doesn't account for Borsh serialization overhead, leading to unnecessary vector reallocations.

In the `create_attestation` instruction, the code constructs a vector with a capacity that underestimates the actual serialized data size:

```
let mut data = Vec::with_capacity(8 + session_address.len() + 32);
data.extend_from_slice(sighash);

let mut session_vec = session_address
    .clone()
    .try_to_vec()
    .map_err(|_| error::ErrorCode::SerializationError)?;
data.append(&mut session_vec);
```

The capacity calculation assumes `session_address.len()` bytes are needed for the string, but Borsh serialization encodes strings with additional metadata. For a string of length `n`, Borsh serialization produces `4 + n` bytes in the format `[length_u32_le][string_bytes]`.

For example:

- A 5-character string like "hello"
- Expected capacity: `8 + 5 + 32 = 45` bytes
- Actual Borsh encoding: `[5, 0, 0, 0, 104, 101, 108, 108, 111]` = 9 bytes
- Total actual size: `8 + 9 + 32 = 49` bytes

The same issue occurs in the `close_attestation` instruction:

```
let mut data = Vec::with_capacity(8 + session_address.len());
data.extend_from_slice(sighash);

let mut session_vec = session_address
    .clone()
    .try_to_vec()
    .map_err(|_| error::ErrorCode::SerializationError)?;
data.append(&mut session_vec);
```

The vector may need to reallocate memory during the `append` operation, negating the optimization benefits.

Recommendations:

Consider pre-serializing the session address string first, then use the actual serialized length for capacity calculation. This ensures the vector is allocated with the correct capacity from the start, preserving the intended optimization benefits.

Alien: Resolved with PR-2

Zenith: Verified.