

LAYERZERO · SECURITY REVIEW

LayerZero

OneSig on Solana.

A security review of OneSig on Solana.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
LayerZero	OneSig on Solana	Ali Shehab, Jakub Zmyslowski, Zdravko Hristov	June 11 – August 16, 2026	August 16, 2026

§ 00

Contents.

39 pages: overview, confidence ranking, scope, 27 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
§ 04	Findings	P. 06
	Main Review 1 · 8 findings	P. 08
	Remediation Review 1 · 2 findings	P. 18
	Main Review 2 · 15 findings	P. 20
	Remediation Review 2 · 2 findings	P. 36
§ 05	Disclaimer	P. 38
§ 06	About Guardian	P. 39

Overview.

Guardian reviewed OneSig on Solana across 5 review rounds. The final report includes 3 Low, 24 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299, https://github.com/GuardianOrg/audit-external-team1-1783712973565
CLIENT	
LayerZero	
PROJECT	METHODOLOGY
OneSig on Solana	Smart contract · Manual · Main Review 1 · Remediation Review 1 · Main Review 2 · Remediation Review 2 · Remediation Review 3
PREPARED BY	
Ali Shehab, Jakub Zmyslowski, Zdravko Hristov	
AUDIT WINDOW	MAIN REVIEW 1 COMMIT
June 11 – August 16, 2026	facaf223b75c
FINAL REPORT	REMEDIATION REVIEW 1 COMMIT
August 16, 2026	c73798b9a6db
	MAIN REVIEW 2 COMMIT
	662a11b7e41a
	REMEDIATION REVIEW 2 COMMIT
	c0197e481012

1.1 Confidence Ranking

5 / 5

Very High Confidence

Guardian assigns a Confidence Ranking of 5 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
Critical	0	0	0	0	0
High	0	0	0	0	0
Medium	0	0	0	0	0
Low	3	0	0	0	3
Informational	24	0	8	0	16
All severities	27	0	8	0	19

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence ASSIGNED	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed OneSig on Solana in

<https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299>, <https://github.com/GuardianOrg/audit-external-team1-1783712973565>.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW 1					
I-01	EmptyExecutorSet check: EVM/Stellar mismatch	Unexpected Behavior	INFORMATIONAL	RESOLVED	p. 8
I-02	Document Solana Signer Caps	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 9
I-03	Root PDA Omits Expiry	Validation	INFORMATIONAL	ACKNOWLEDGED	p. 10
I-04	Inline verification ignores merkle_root_state	Best Practices	INFORMATIONAL	ACKNOWLEDGED	p. 11
I-05	Misleading "permissionless" doc comment	Documentation	INFORMATIONAL	RESOLVED	p. 13
I-06	PDA Signer Propagates Through CPI	Documentation	INFORMATIONAL	ACKNOWLEDGED	p. 15
I-07	Solana Leaf Is Single-Instruction	Informational	INFORMATIONAL	RESOLVED	p. 16
I-08	Undocumented Solana per-leaf tx-size limit	Best Practices	INFORMATIONAL	ACKNOWLEDGED	p. 17
REMEDATION REVIEW 1					
I-01	Stale build_instruction doc comment	Documentation	INFORMATIONAL	ACKNOWLEDGED	p. 18
I-02	Cached Signer Subset Can Be Front-Run	Frontrunning	INFORMATIONAL	ACKNOWLEDGED	p. 19
MAIN REVIEW 2					
L-01	Mixed writability breaks multicalls	Logical Error	LOW	RESOLVED	p. 20
L-02	Foreign buffers can be consumed	DoS	LOW	RESOLVED	p. 22
L-03	Delegate authorization omits seed	Signatures	LOW	RESOLVED	p. 23
I-01	Non-unique one_sig_id can confuse indexing	Validation	INFORMATIONAL	RESOLVED	p. 24
I-02	Expiry uint256 encoding lacks documentation	Documentation	INFORMATIONAL	RESOLVED	p. 25
I-03	Ambiguous seed documentation	Documentation	INFORMATIONAL	RESOLVED	p. 26
I-04	Misleading domain separator comment	Documentation	INFORMATIONAL	RESOLVED	p. 27
I-05	Cross-state OneSig execution is rejected	Unexpected Behavior	INFORMATIONAL	ACKNOWLEDGED	p. 28
I-06	SOL allowance documents wrong limit	Documentation	INFORMATIONAL	RESOLVED	p. 29
I-07	Unused signer authorization is ignored	Documentation	INFORMATIONAL	RESOLVED	p. 30
I-08	Executor policy is not signed	Documentation	INFORMATIONAL	RESOLVED	p. 31

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-09	SVM accepts non-EVM recovery ids	Signatures	INFORMATIONAL	RESOLVED	p. 32
I-10	Unlimited SOL allowance may overflow	Documentation	INFORMATIONAL	RESOLVED	p. 33
I-11	PDA authority trust is underdocumented	Documentation	INFORMATIONAL	RESOLVED	p. 34
I-12	Buffered close events are missing	Events	INFORMATIONAL	RESOLVED	p. 35

REMEDIATION REVIEW 2

I-01	Document one_sig_id chain-id invariant	Documentation	INFORMATIONAL	RESOLVED	p. 36
I-02	Close events missing from docs	Documentation	INFORMATIONAL	RESOLVED	p. 37

INFORMATIONAL I-01

RESOLVED

EmptyExecutorSet check: EVM/Stellar mismatch

CATEGORY	Unexpected Behavior
LOCATION	packages/onesig/onesig-solana/programs/onesig/src/state.rs
REVIEW	Main Review 1

DESCRIPTION

`Executors::set_executor_required` (and, transitively, `init_one_sig`) and `Executors::remove_executor` enforce the invariant that the executor set must be non-empty whenever `executor_required = true`, reverting with `OneSigError::EmptyExecutorSet` otherwise. This is stricter than the EVM and Stellar implementations, which both allow `executor_required = true` with an empty executor set. The invariant is unnecessary because execution does not depend on the executor set being non-empty. When `executor_required = true`, a registered signer can still execute through `signer_execute_transaction`, which validates the submission against `one_sig_state.multisig.signers` (the signer set) and never reads `executors`. Since `threshold >= 1` always guarantees at least one signer, the state `(executor_required = true, executors = [])` is fully functional, it simply disables the standalone executor path while leaving the signer path available. It is not a lockout. The practical effect is a cross-VM behavioral mismatch: a valid "signers-only, no standalone executors" configuration that is representable on EVM and Stellar cannot be configured on Solana, because the guard rejects it at both `set_config` and `init`.

RECOMMENDATION

Consider dropping the non-empty executor-set requirement so `executor_required = true` is permitted with an empty executor set, matching EVM and Stellar. Or acknowledging and clearly documenting this mismatch.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-02

ACKNOWLEDGED

Document Solana Signer Caps

CATEGORY	Informational
LOCATION	packages/onesig/onesig-solana/programs/onesig/src/constants.rs:21
REVIEW	Main Review 1

DESCRIPTION

The Solana implementation intentionally caps each OneSig signer set at 20 signers and caps the signature threshold at 13. These limits are enforced during initialization and config updates: `add_signer` rejects more than `SIGNERS_MAX_LEN`, and `set_threshold` rejects 0, values above `MAX_THRESHOLD`, and thresholds greater than the current signer count.

This appears intentional and is documented in code comments as a Solana transaction-size constraint, not a direct vulnerability.

However, it is important operationally because other implementations, such as EVM, do not have the same explicit 20 signer / 13 threshold cap. A cross-chain deployment that assumes the same governance policy can be mirrored everywhere may fail on Solana, or may require a lower Solana threshold than the rest of the deployment.

Operators should also be aware that having 20 signers does not mean a 20-of-20 policy is possible on Solana; the maximum threshold remains 13.

RECOMMENDATION

Document the Solana-specific signer and threshold caps in public integration docs, deployment tooling, and operator runbooks. Off-chain tooling should validate these limits before producing cross-chain signer-set ceremonies, and should warn when an

EVM/Stellar/etc. signer policy cannot be represented equivalently on Solana.

For Solana execution, tooling should also prefer supplying only the required threshold number of signatures to avoid unnecessary transaction-size and compute pressure.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-03

ACKNOWLEDGED

Root PDA Omits Expiry

CATEGORY	Validation
LOCATION	packages/onesig/onesig-solana/programs/onesig/src/instructions/verify_merkle_root.rs:15
REVIEW	Main Review 1

DESCRIPTION

`verify_merkle_root` derives `MerkleRootState` only from `[MERKLE_ROOT_SEED, one_sig_state, merkle_root]`, while expiry is only stored inside the account. Since expiry is part of the signed EIP-712 payload, the same Merkle root may validly be signed with a different expiry. However, once a PDA exists for that root, another verification for the same root with a new expiry cannot initialize a fresh `MerkleRootState`. This means a root verified with a shorter expiry can occupy the only PDA slot for that root. A later valid signature set extending the same root's expiry cannot be cached through the two-step path until the original PDA expires and is closed.

One-step execution with inline signatures remains possible, so this is not a full execution liveness break, but it can break the intended pre-verification flow and create an avoidable griefing/availability issue.

EVM does not have the same issue because it does not persist verified roots; it verifies (merkleRoot, expiry, signatures) live on every execution.

RECOMMENDATION

Include expiry in the `MerkleRootState` PDA seeds, or allow `verify_merkle_root` to update an existing PDA when the caller supplies a valid signature set for the same root with a later expiry.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-04

ACKNOWLEDGED

Inline verification ignores merkle_root_state

CATEGORY Best Practices

LOCATION packages/onesig/onesig-solana/programs/onesig/src/execution.rs

REVIEW Main Review 1

DESCRIPTION

`resolve_merkle_root` (`execution.rs`) branches on `merkle_root_verification`:

```

`rust
let root = if let Some(VerifyMerkleRootParams { merkle_root, expiry, signatures }) =
merkle_root_verification
{
// Case 1: Direct verification with merkle root parameters
MerkleValidator::verify_merkle_root(
one_sig_state, merkle_root, *expiry, signatures.as_ref(), Clock::get()?.unix_timestamp,
)?;
*merkle_root
} else {
// Case 2: Two-step verification, using pre-verified merkle root state
require!(merkle_root_state.is_some(), OneSigError::MissingMerkleRootState);
merkle_root_state.unwrap().merkle_root
};
`

```

The two modes are mutually exclusive by construction: `inline ( Some )` verifies the root from the params and returns it; `two-step ( else )` requires the pre-verified account. The `inline` branch never reads `merkle_root_state`, so the intended pairing is `inline  $\Rightarrow$  merkle_root_state == None`, `two-step  $\Rightarrow$  merkle_root_state == Some`.

Nothing enforces this. A caller of either entrypoint can supply both `merkle_root_verification == Some(...)` and a populated `merkle_root_state`; the `inline` branch wins and the account is silently ignored as contradictory dead input. The issue is that the program tolerates self-contradictory input instead of rejecting it, an inconsistency between the invariant the code is built around (`inline  $\Rightarrow$  no stored account`) and what is actually enforced (nothing).

I-04 Inline verification ignores merkle_root_state

CONTINUED

RECOMMENDATION

Reject a co-supplied account in the inline branch:

```
``rust
require!(merkle_root_state.is_none(), OneSigError::UnexpectedMerkleRootState);
``
```

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-05

RESOLVED

Misleading "permissionless" doc comment

CATEGORY Documentation

LOCATION packages/onesig/onesig-solana/programs/onesig/src/instructions/execute_transaction.rs

REVIEW Main Review 1

DESCRIPTION

The doc comment on `ExecuteTransaction::apply` (`instructions/execute_transaction.rs:33`) reads:

```
``rust
/// Executes a transaction with a pre-verified merkle root permissionlessly
`
This is only true when executor_required == false. When executor_required == true, the path is
executor-gated, not permissionless: apply requires the executor (a Signer) to be present in
OneSigState.executors.executors, otherwise it aborts with OneSigError::ExecutorRequired:
`rust
if ctx.accounts.one_sig_state.executors.executor_required {
let executor = ctx.accounts.executor.key();
require!(
ctx.accounts.one_sig_state.executors.executors.contains(&executor),
OneSigError::ExecutorRequired
);
}
```

So in the executor-required configuration, only a pre-approved executor can drive this instruction, the call is permissioned. The gating logic itself is correct; only the comment is wrong. The same caveat applies to the "1. Verify signatures ..." process note that follows, which describes the permissionless flow without mentioning the executor check that runs first.

RECOMMENDATION

Correct the comment to reflect both modes; for example, note that execution is permissionless only when `executor_required == false`, and is restricted to an approved executor in `OneSigState.executors` when `executor_required == true`.

I-05 Misleading "permissionless" doc comment

CONTINUED

— **RESOLUTION**

Status: Resolved.

INFORMATIONAL I-06

ACKNOWLEDGED

PDA Signer Propagates Through CPI

CATEGORY	Documentation
LOCATION	Global
REVIEW	Main Review 1

— DESCRIPTION

Solana OneSig executes target instructions through `invoke_signed`, marking `one_sig_signer` as signer for the target CPI whenever the PDA is included in the signed leaf's account list. On Solana, signer privileges are inherited by downstream CPIs, so the immediate target program can pass `one_sig_signer` as signer to other programs using the accounts it received.

This differs from the EVM implementation. EVM OneSig uses normal `.call`, so a downstream contract called by the target sees `msg.sender == target`, not `msg.sender == OneSig`. In contrast, Solana signer privilege can continue through nested CPI. Solana runtime prevents indirect reentry back into OneSig, so this does not allow a target to call back into OneSig admin functions such as `set_config`, but it does mean that any target receiving `one_sig_signer` must be trusted to use that signer authority correctly for all downstream CPIs.

— RECOMMENDATION

Document this Solana-specific trust boundary. Off-chain tooling should make it explicit when a signed leaf grants `one_sig_signer` signer authority to a target program. Operators should treat target programs that receive `one_sig_signer` as trusted for the full downstream CPI flow, not only for the immediate instruction.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-07

RESOLVED

Solana Leaf Is Single-Instruction

CATEGORY Informational

LOCATION packages/onesig/onesig-solana/programs/onesig/src/execution.rs:52

REVIEW Main Review 1

— DESCRIPTION

This is the semantic/model mismatch.

- EVM leaf: contains Call\[\]
- Solana leaf: contains one target instruction
- Multi-call workflows need multiple leaves or a wrapper program

— RECOMMENDATION

Beware of this constraint.

— RESOLUTION

Status: Resolved.

INFORMATIONAL I-08

ACKNOWLEDGED

Undocumented Solana per-leaf tx-size limit

CATEGORY	Best Practices
LOCATION	packages/onesig/onesig-solana/programs/onesig/src/validation/merkle.rs
REVIEW	Main Review 1

— DESCRIPTION

Solana OneSig signers can authorize a leaf that is valid under the Merkle/signature scheme but cannot be submitted because the corresponding `execute_transaction` exceeds Solana transaction-size or account/compute limits. In the two-step path, each execution still carries the target instruction data, target accounts, and Merkle proof. Address Lookup Tables reduce account-address overhead, but they do not reduce instruction data or proof bytes. Large roots increase proof size logarithmically, reducing the remaining budget for the target instruction.

If an unexecutable leaf corresponds to the current nonce, later leaves that commit to higher nonces cannot execute through the normal sequence until signers authorize a different executable leaf for that nonce.

— RECOMMENDATION

Consider adding SDK/tooling preflight checks that estimate the serialized transaction size for every Solana leaf under its expected proof depth and account list. Tooling should warn or reject leaves that cannot fit even with ALTs and two-step verification, and should recommend splitting roots, reducing proof depth, reducing instruction data/accounts, or using a wrapper program where appropriate.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-01

ACKNOWLEDGED

Stale build_instruction doc comment

CATEGORY Documentation

LOCATION packages/onesig/onesig-solana/programs/onesig/src/execution.rs:48

REVIEW Remediation Review 1

DESCRIPTION

The doc comment on `build_instruction` (execution.rs:48-53) is out of date relative to the function it documents. After this rem, `build_instruction` now (a) returns `Result<OneSigInstruction>` instead of `OneSigInstruction`, and (b) guards against an empty `remaining_accounts` slice via `require!(!remaining_accounts.is_empty(), OneSigError::MissingProgramId)` before indexing `remaining_accounts[0]`.

The comment still describes the old behavior: it lists "5. Returns the `OneSigInstruction`" (no mention of the fallible `Result`) and step "1. Calculates the start and end indices for accounts" / "3. Creates an instruction with the program ID from the first account" without noting the new empty-accounts precondition that protects that first-element access. A reader relying on the doc would not learn that the function can now fail with `MissingProgramId`, nor that the empty-input panic was fixed here. The code itself is correct; only the comment is stale

RECOMMENDATION

Update the comment to reflect the current signature and behavior: note that the function returns `Result<OneSigInstruction>` and errors with `OneSigError::MissingProgramId` when `remaining_accounts` is empty (before the `remaining_accounts[0]` program-id read).

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-02

ACKNOWLEDGED

Cached Signer Subset Can Be Front-Run

CATEGORY	Frontrunning
LOCATION	packages/onesig/onesig-solana/programs/onesig/src/instructions/verify_merkle_root.rs
REVIEW	Remediation Review 1

DESCRIPTION

The I-01 remediation stores the signers supplied during `verify_merkle_root` in `MerkleRootState.signed_by`. Subsequent execution through the two-step flow requires a sufficient number of **those specific cached signers** to remain active.

Because Merkle root verification is permissionless and each Merkle root is associated with a unique PDA, an attacker can front-run a legitimate verification transaction by copying it, removing any unnecessary signatures, and initializing the PDA using only the minimum quorum required for verification.

For example, if the signature threshold is **2** and the original verification includes signatures from **A**, **B**, and **C**, a front-runner can submit the same Merkle root using only **A** and **B**. The PDA is then permanently initialized with `{A, B}` as the cached signer set. If signer **A** is later removed, the cached root can no longer satisfy execution requirements through the two-step path, even though **B** and **C** still form a valid active quorum under the current signer configuration.

Since the PDA cannot be updated or reverified, the legitimate verification containing the stronger signer snapshot (`{A, B, C}`) cannot replace the front-runner's minimal subset until the account becomes closeable. As a result, the cached Merkle root remains unusable for two-step execution despite the existence of a valid active quorum.

Inline Merkle proof verification remains unaffected, making this primarily an availability and griefing issue that impacts the two-step execution flow rather than enabling unauthorized execution.

RECOMMENDATION

Beware of this case when using pre-verified roots. If you intend to resolve it, consider adding a method that allows an existing root PDA to be refreshed with a newly verified signer set.

RESOLUTION

Status: Acknowledged.

LOW **L-01**

RESOLVED

Mixed writability breaks multicalls

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L256-L265 https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/src/onesig.ts#L254-L276
REVIEW	Main Review 2

DESCRIPTION

Multicalls reference accounts only by index and do not encode the writability expected by each individual call. During reconstruction, the program copies the privilege with which the account was supplied to `execute_transaction` from `AccountInfo.is_writable`, rather than using the writability intended for the individual inner call.

```
``plain text
for &index in instruction_account_indices {
let account_info = account_at(index)?;
let pubkey = account_info.key();
metas.push(OneSigAccountMeta {
pubkey,
is_signer: pubkey == one_sig_signer_key,
is_writable: account_info.is_writable,
});
}
```

Consider an ordered two-call leaf where both calls use account X. Call A expects X to be read-only, while Call B expects X to be writable. Because Call B requires write access, account X must be supplied as writable to the OneSig `execute_transaction` instruction. As a result, `AccountInfo.is_writable` is true when OneSig reconstructs both inner calls, even though Call A was signed with X as read-only.

The signed Merkle root commits to a leaf containing X as read-only in Call A and writable in Call B, but the program reconstructs X as writable in both calls during execution. Since writability is included in `OneSigInstruction`, the reconstructed leaf hash differs from the signed leaf, causing Merkle-proof verification to fail before either CPI executes.

RECOMMENDATION

Encode the expected writability for every account reference in each call instead of reconstructing it from `AccountInfo.is_writable`. During reconstruction, use the per-call flag and reject writable access if the account was supplied as read-only.

L-01 Mixed writability breaks multicalls

CONTINUED

— **RESOLUTION**

Status: Resolved.

LOW **L-02**

RESOLVED

Foreign buffers can be consumed

CATEGORY DoS

LOCATION `init_buffered_transaction.rs#L35-L41` (https://github.com/Guardian0rg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/init_buffered_transaction.rs#L35-L41) `init_buffered_signatures.rs#L34-L39` (https://github.com/Guardian0rg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/init_buffered_signatures.rs#L34-L39)

REVIEW Main Review 2

DESCRIPTION

`init_buffered_transaction()` and `init_buffered_signatures()` derive their buffered account PDAs from the intended `one_sig_state`, `params.merkle_root`, and `authority` values.

`init_buffered_transaction()` also includes `params.nonce`. However, `execute_transaction()` accepts `buffered_transaction` and `buffered_signatures` accounts without revalidating that their PDA seeds match the executing `OneSigState`, `merkle_root`, `nonce`, or `authority`.

As a result, a buffered account created for one `OneSigState` can be supplied during execution against another `OneSigState` if the attacker can make the buffered payload or signatures verify under the attacker-controlled execution context. For `BufferedTransaction`, successful execution closes the account unconditionally, deleting the victim's staged transaction and forcing it to be re-uploaded. For `BufferedSignatures`, successful execution can close the account when the executing `one_sig_state.nonce` exceeds the stored `closable_after_nonce`, so an attacker-controlled state with a sufficiently advanced nonce can consume staged signatures as well.

RECOMMENDATION

In `execute_transaction()`, derive the expected PDA using the `authority` read from the provided buffered account together with the executing context seeds, then require the provided account key to match it. For `BufferedTransaction`, derive the PDA from `BUFFERED_TRANSACTION_SEED`, the current `one_sig_state`, `merkle_root`, current nonce, and `buffered_transaction.authority`. For `BufferedSignatures`, derive the PDA from `BUFFERED_SIGNATURES_SEED`, the current `one_sig_state`, `merkle_root`, and `buffered_signatures.authority`.

RESOLUTION

Status: Resolved.

LOW **L-03**

RESOLVED

Delegate authorization omits seed

CATEGORY Signatures

LOCATION `signature.rs#L118-L124` (<https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/validation/signature.rs#L118-L124>)

REVIEW Main Review 2

DESCRIPTION

`build_signer_execution_authorization_digest()` signs only the `leaf`, `merkle_root`, `delegate`, and `expiry`. Unlike root signatures, delegate execution authorizations are not bound to the current `seed`. As a result, after `seed` rotation invalidates old root signatures, an old delegate authorization can still be reused if the same `merkle_root` is signed again under the new `seed`.

RECOMMENDATION

Include the current `seed` in the signed `SignerExecutionAuthorization` data and pass it into `build_signer_execution_authorization_digest()`, so delegate authorizations are invalidated together with root signatures when `seed` changes.

RESOLUTION

Status: Resolved.

Guardian: The remediation response for `[**Delegate authorization omits seed**]` ([/p/39d8bda5828c81dca650e0d6590ac2ad?v=fd7bbb7870f543fc942bdbfb735f3899&pvs=25](https://p/39d8bda5828c81dca650e0d6590ac2ad?v=fd7bbb7870f543fc942bdbfb735f3899&pvs=25)) states that omitting `seed` from `SignerExecutionAuthorization` is intentional and that the behavior would be documented. The current signer authorization documentation explains when `signer_authorization` is verified or ignored, but it does not document that `SignerExecutionAuthorization` intentionally omits `seed`, that `seed` is enforced through root-signature verification, or that re-approving the same `merkleRoot` after a `seed` rotation preserves existing unexpired delegate authorizations for that root.

****Recommendation****

Document the intended split between root validity and delegate authorization: `seed` controls whether a `merkleRoot` is valid, while `SignerExecutionAuthorization` authorizes a `delegate` for a specific `leafHash` and `merkleRoot`. Also document that if signers re-approve the same `merkleRoot` after a `seed` rotation, existing unexpired delegate authorizations for that root remain valid.

INFORMATIONAL I-01

RESOLVED

Non-unique one_sig_id can confuse indexing

CATEGORY Validation

LOCATION `init_one_sig.rs#L14` (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/init_one_sig.rs#L14)

REVIEW Main Review 2

DESCRIPTION

`init_one_sig()` accepts a caller-provided `one_sig_id` and `InitOneSig::apply()` stores it directly in `OneSigState` without enforcing uniqueness. The `state` account is initialized as a normal caller-provided account rather than a PDA derived from `one_sig_id`, and the `one_sig_signer` PDA is derived from `ONE_SIG_SEED` and `state.key()`. As a result, multiple unrelated `OneSigState` accounts can share the same `one_sig_id`.

RECOMMENDATION

Document that `one_sig_id` is not globally unique and that the canonical Solana instance identifier is the `OneSigState` pubkey. If uniqueness is intended, derive `state` from `one_sig_id` with PDA seeds or introduce a registry that reserves each `one_sig_id`, with appropriate anti-squatting controls for permissionless initialization.

Also consider using `one_sig_id` for chain separation if this is not already the case - for example, not letting a leaf from one EVM chain to be executed on another.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-02

RESOLVED

Expiry uint256 encoding lacks documentation

CATEGORY	Documentation
LOCATION	merkle.rs#L120-L121 (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/validation/merkle.rs#L120-L121)
REVIEW	Main Review 2

DESCRIPTION

Solana accepts `expiry` as a `u64` for both root execution and signer-executor authorization, while the corresponding EIP-712 digests encode `expiry` into a 32-byte `uint256` field. This is implemented correctly, and the Solana SDK handles the bound by serializing both `ExecuteTransactionInstructionData.expiry` and `SignerAuthorization.expiry` with `u64()`, which rejects values above `u64::MAX` before instruction construction. However, the Solana-facing documentation does not clearly explain that `expiry` is only represented as `uint256` inside the EIP-712 signing preimage, while the valid Solana input domain remains Unix-seconds timestamps representable as `u64`. Integrators constructing signatures or digests outside the standard Solana SDK path may miss this distinction.

RECOMMENDATION

Document the Solana `expiry` semantics for both root signatures and signer-executor authorizations: `expiry` must be a Unix-seconds timestamp that fits in `u64`, and it is left-padded into a 32-byte EIP-712 `uint256` slot only when computing the signing digest.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-03

RESOLVED

Ambiguous seed documentation

CATEGORY	Documentation
LOCATION	init_one_sig.rs#L15-L18 (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/init_one_sig.rs#L15-L18)
REVIEW	Main Review 2

DESCRIPTION

The comment above `InitOneSigParams::seed` says `*seed*` is the only cross-VM instance selector*, which may be understood as if the `seed` determines the VM the txs will be executed on. This is not the role of the seed, as it is signed together with the merkle root.

RECOMMENDATION

Consider whether the comment needs to be modified.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-04

RESOLVED

Misleading domain separator comment

CATEGORY Documentation

LOCATION `constants.rs#L52-L58` (<https://github.com/Guardian0rg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/constants.rs#L52-L58>)

REVIEW Main Review 2

DESCRIPTION

The comment for `DOMAIN_SEPARATOR` states that the EIP-191 `0x1901` prefix is included when computing the domain separator. This is inaccurate: `DOMAIN_SEPARATOR` is correctly computed as the EIP-712 domain separator only. The EIP-191 prefix is applied later when constructing the final signing digest in `build_merkle_root_digest()` and `build_signer_execution_authorization_digest()`.

RECOMMENDATION

Remove the EIP-191 prefix bullet from the `DOMAIN_SEPARATOR` comment so the documentation matches the implemented EIP-712 digest construction.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-05

ACKNOWLEDGED

Cross-state OneSig execution is rejected

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L321-L324
REVIEW	Main Review 2

DESCRIPTION

The reentrancy guard rejects every inner call to `execute_transaction` targeting the OneSig program, without distinguishing between recursion into the currently executing state and execution through an independent OneSig state:

```
`plain text
if solana_ix.program_id == ID {
let exec_disc = crate::instruction::ExecuteTransaction::DISCRIMINATOR;
require(!solana_ix.data.starts_with(exec_disc), OneSigError::Reentrancy);
}
`
```

Multiple independent `OneSigState` accounts can exist under the same program deployment. Consider states A and B, where B allowlists A's signer PDA as an executor. Because the PDA can only sign through CPI, A must invoke B using `invoke_signed`.

Although B would independently validate its executor, Merkle root, signatures, proof, and nonce, the guard rejects the call solely because it targets the OneSig program, without checking that the target is a different state.

As a result, one OneSig instance cannot execute through another, even when authorized, what blocks valid cross-instance calls.

RECOMMENDATION

Consider scoping the reentrancy protection to the currently executing `OneSigState`. If all cross-state execution is intentionally unsupported, document this restriction explicitly.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-06

RESOLVED

SOL allowance documents wrong limit

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L332-L336 https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/types.rs#L172
REVIEW	Main Review 2

DESCRIPTION

The `value` field is documented as the maximum amount of SOL that a call may spend. However, the implementation does not measure the total amount transferred from the OneSig signer PDA. It only compares the signer's balance before and after the CPI.

For example, assume the signer begins with 10 SOL, receives 100 SOL during the CPI, transfers out 105 SOL, and finishes with 5 SOL. With value set to 5 SOL, the check passes because the signer's final net loss is only 5 SOL, even though 105 SOL was transferred out.

This may cause integrators to rely on a spending restriction that the implementation does not enforce.

RECOMMENDATION

Document value as the maximum permitted net decrease in the signer PDA's lamport balance.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-07

RESOLVED

Unused signer authorization is ignored

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L286-L288
REVIEW	Main Review 2

DESCRIPTION

When `executor_required` is disabled, execution is permissionless and any supplied `signer_authorization` is ignored. The same applies when the executor is already allowlisted.

Consequently, a successful transaction does not prove that the supplied authorization was validated or valid.

Although this has no on-chain authorization impact, integrations may incorrectly interpret the presence of the field as evidence that its signature and expiry were validated.

RECOMMENDATION

Explicitly document in the public SDK that `signerExecutionSignature` and its expiry are ignored when execution is permissionless or the executor is already allowlisted.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-08

RESOLVED

Executor policy is not signed

CATEGORY	Documentation
LOCATION	<code>execute_transaction.rs#L286-L287</code> (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L286-L287)
REVIEW	Main Review 2

DESCRIPTION

`execute_transaction()` enforces executor authorization from the current `OneSigState` configuration. The signed Merkle root commits to `seed`, `merkle_root`, and `expiry`, while each leaf commits to the target state, nonce, and call data. Neither the root nor the leaf commits to `executor_required` or the executor allowlist.

As a result, an already-signed root can be executed under a different executor policy than the one operators expected at signing time. For example, if `executor_required` is later disabled while an old root is still valid and nonce-compatible, leaves from that root become permissionlessly executable.

RECOMMENDATION

Document that executor policy is live `OneSigState` configuration and is not part of the signed root or leaf. If executor policy is intended to be fixed at signing time, include it in the signed data or invalidate outstanding roots when executor configuration changes.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-09

RESOLVED

SVM accepts non-EVM recovery ids

CATEGORY	Signatures
LOCATION	signature.rs#L81-L84 (https://github.com/Guardian0rg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/validation/signature.rs#L81-L84)
REVIEW	Main Review 2

DESCRIPTION

`recover_signer()` normalizes any `recovery_id` in `27..=30` by subtracting `27`, and otherwise forwards the raw `recovery_id` to `secp256k1_recover()`. This means Solana accepts signature encodings with raw `v` values `0/1`, and can also accept valid raw `v` values `2/3` or their `29/30` equivalents when `secp256k1_recover()` successfully recovers the signer.

This differs from the EVM path using OpenZeppelin-style `ECDSA.tryRecover()`, which accepts only `v = 27/28` for 65-byte signatures. As a result, the same signed root or executor delegate can be accepted on SVM with a recovery-id encoding that would be rejected on EVM.

While this issue doesn't lead to direct exploitation of the system, it's of the same class as [Permissive secp256k1 Recovery](https://app.notion.com/p/guardianaudits/I-02-37c8bda5828c80ab8e26e73741f793a8?v=37b8bda5828c8126bcab000cc65c7c29&source=copy_link) because it allows multiple variants of the same signature to be used.

Also, the implementation accepts `2` and `3` values which are normally not expected for Ethereum-style signed messages.

RECOMMENDATION

Reconsider whether this flexibility is desired or it's better to stick to only one variant.

RESOLUTION

Status: Resolved.

Guardian: The latest change in `signature.rs` removes support for `29..=30` `recovery_id` values, but the underlying function still accepts raw `0/1` which is not documented in any user-facing docs.

****Recommendation****

Consider documenting the fact that both `0/1`, `27/28` can be used for `recovery_id`.

INFORMATIONAL I-10

RESOLVED

Unlimited SOL allowance may overflow

CATEGORY Documentation

LOCATION https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L333-L336

REVIEW Main Review 2

DESCRIPTION

The program verifies each call's SOL allowance by adding the approved value to the signer's post-execution balance.

```
``plain text
require!(
  balance_before <= balance_after + value,
  OneSigError::ExcessiveBalanceDeduction
);
`
```

When value is `u64::MAX` and `balance_after`` is nonzero, this addition overflows. Because release builds enable overflow checks, the transaction aborts and all CPI effects are rolled back.

Integrators may interpret the maximum `u64` value as an unlimited SOL spending allowance. Consequently, this assumption can make a valid authorized call unexecutable.

RECOMMENDATION

Document that value is a finite allowance and that `u64::MAX` must not be treated as an unlimited approval.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-11

RESOLVED

PDA authority trust is underdocumented

CATEGORY	Documentation
LOCATION	<code>execute_transaction.rs#L326-L330</code> (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L326-L330)
REVIEW	Main Review 2

DESCRIPTION

OneSig executes approved Solana calls by invoking the target program with `invoke_signed()`, which grants the `one_sig_signer` PDA signer authority to the callee for the duration of the CPI. The signed leaf commits to the target `program_id`, account metas, instruction data, lamport allowance, nonce, `one_sig_id`, and OneSig state, but it does not commit to the target program's code, `ProgramData`, or upgrade authority. As a result, if signers approve a call to an upgradeable target and that target is upgraded before execution, the new code can still receive the PDA signer authority over the signed account set. In addition, when `one_sig_signer` is passed as writable, a target can temporarily initialize, assign, or otherwise use the PDA during nested CPIs and restore it to a system-owned empty account before returning, satisfying OneSig's post-CPI checks. This is consistent with Solana CPI semantics, but the trust assumption is not clearly documented for signers and integrators.

This can be combined with [SOL allowance documents wrong limit]

(/p/39d8bda5828c81e1b7faf99c3e403ed1?v=fd7bbb7870f543fc942bdbfb735f3899&pvs=25) if there is value to be claimed on behalf of the PDA.

RECOMMENDATION

Document that OneSig signatures authorize execution against the target `program_id` and signed account set at execution time, not against immutable target code semantics. The documentation should warn signers to account for target upgradeability, target upgrade authority, and writable `one_sig_signer` usage before signing; prefer frozen or trusted target programs, short expiries, and avoid passing `one_sig_signer` as writable unless the intended call requires it.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-12

RESOLVED

Buffered close events are missing

CATEGORY	Events
LOCATION	<code>execute_transaction.rs#L364-L385</code> (https://github.com/GuardianOrg/audit-external-team1-1783712973565/blob/662a11b7e41a796a6f270ee232f5975143558b9c/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/execute_transaction.rs#L364-L385)
REVIEW	Main Review 2

DESCRIPTION

Buffered account lifecycle events are emitted when `BufferedSignatures` and `BufferedTransaction` accounts are initialized, but no corresponding close events are emitted when those accounts are closed. This includes both manual close paths and the automatic cleanup paths in `maybe_close_buffered_signatures()` and `maybe_close_buffered_transaction()`. As a result, off-chain monitors and indexers can observe `BufferedSignaturesInitialized` and `BufferedTransactionInitialized`, but must infer closure by inspecting account state, lamport movement, or transaction effects rather than consuming an explicit lifecycle event.

RECOMMENDATION

Consider emitting explicit close events for `BufferedSignatures` and `BufferedTransaction` accounts from both manual close instructions and automatic cleanup in `execute_transaction()`.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-01

RESOLVED

Document one_sig_id chain-id invariant

CATEGORY	Documentation
LOCATION	<code>init_one_sig.rs#L13-L15</code> (https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299/blob/c0197e4810124d2a6fa652ce208dd47913662d78/packages/onesig/onesig-solana-staged/programs/onesig/src/instructions/init_one_sig.rs#L13-L15) <code>onesig-leaf.ts#L5-L7</code> (https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299/blob/c0197e4810124d2a6fa652ce208dd47913662d78/packages/onesig/onesig-model/src/model/onesig-bundle/onesig-leaf.ts#L5-L7) <code>index.ts#L84-L90</code> (https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299/blob/c0197e4810124d2a6fa652ce208dd47913662d78/packages/onesig/onesig-core/src/index.ts#L84-L90)
REVIEW	Remediation Review 2

DESCRIPTION

Solana documentation for `one_sig_id` has been revised to clarify that it is caller-provided and not globally unique. However, the broader `OneSig` codebase still assumes `oneSigId` represents the chain EID: the shared leaf schema describes `oneSigId` as the EID of the chain, and `makeOneSigTree()` treats duplicate `(nonce, oneSigId)` pairs as duplicate transactions for the same chain and nonce. This cross-codebase assumption is not documented consistently in the Solana implementation, making it unclear whether Solana deployments must choose `one_sig_id` as the chain EID or may choose arbitrary application identifiers.

RECOMMENDATION

Document the intended invariant consistently across the Solana implementation and shared `OneSig` model.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-02

RESOLVED

Close events missing from docs

CATEGORY Documentation

LOCATION AUDIT.md#L171 (<https://github.com/GuardianOrg/layerzero-audit-external-source-team1-1781115004299/blob/c0197e4810124d2a6fa652ce208dd47913662d78/packages/onesig/onesig-solana-staged/AUDIT.md#files-in-scope-programsonesigsrc>)

REVIEW Remediation Review 2

DESCRIPTION

The `AUDIT.md` component map lists the events emitted by `events.rs`, but omits the newly added `BufferedSignaturesClosed` and `BufferedTransactionClosed` events. The runtime now emits these events from `close_buffered_signatures()`, `close_buffered_transaction()`, and the buffered auto-close paths in `execute_transaction()`, so the documentation no longer reflects the full event surface.

RECOMMENDATION

Update the `events.rs` entry in `AUDIT.md` to include `BufferedSignaturesClosed` and `BufferedTransactionClosed`.

RESOLUTION

Status: Resolved.

§ 05

Disclaimer.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian's position is that each company and individual are responsible for their own due diligence and continuous security. Guardian's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.