SECURITY ASSESSMENT

Accretion

Provided by Accretion Labs Pte Ltd. for Solana Foundation
February 27, 2026
A26SFR1

Solana Foundation

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	Niklas Brymko (niklas@accretion.xyz)
Auditor	Mahdi Rostami (mahdi@accretion.xyz)

CLIENT

Solana Foundation (https://solana.org) engaged Accretion to conduct a security assessment of the LazorKit smart wallet management system on Solana that provides secure passkey authentication, customizable policy engines, and flexible transaction execution capabilities.

ENGAGEMENT TIMELINE

26 Jan — Project Kickoff
Initial planning and scope definition

26 Jan — Assessment Begins
Security review and testing phase

02 Feb — Review Fixes
Security recommendations are given and implemented

02 Feb — Project Completion
Report delivery and on-chain confirmation

AUDITED CODE

#	Program ID	Repository
1	Gsuz7YcA5sbMGVRXT3xSYhJBessW4xFC4xYsihNCqMFh	https://github.com/lazor-kit/program-v2
2	BiE9vSdz9MidUiyjVYsu3PG4C1fbPZ8CVPADA9jRfXw7	https://github.com/lazor-kit/program-v2

ASSESSMENT

The security assessment of Solana Foundation's LazorKit revealed 14 issues across all severity levels: 2 critical, 2 high, 2 medium, 5 low, and 3 informational. All identified issues have been successfully fixed by the development team, demonstrating strong remediation efforts.

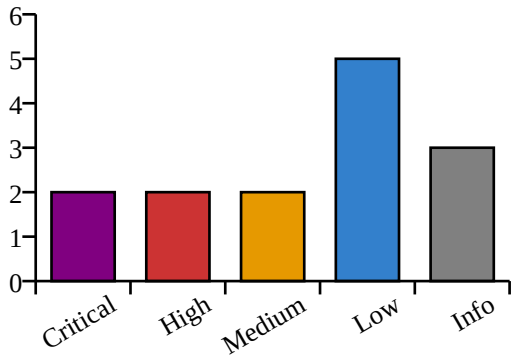
CODE ASSESSMENT

The codebase implements Solana-native smart contracts with multi-signature wallet functionality. Code structure follows standard Rust/Anchor patterns but initially lacked proper input validation and signature binding mechanisms in critical authentication flows.

KEY FINDINGS

Critical issues included cross-wallet authority deletion vulnerability and account reordering attacks due to insufficient payload binding. High-severity findings involved Secp256r1 layout mismatches breaking validation and missing account signatures enabling unauthorized operations and rent theft.

SEVERITY DISTRIBUTION



ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: LazorKit Program
Link: https://github.com/lazor-kit/program-v2
Commit: cd09588d2459571ceb6fe7bd8764abb139b6d3de
Program ID: Gsuz7YcA5sbMGVRXT3xSYhJBessW4xFC4xYsihNCqMFh
Audit Result:
• **Audited Commit:** 79cfc6c3fcd16df63011e507f8da09092c17dc40

Item 2: Default Policy Program
Link: https://github.com/lazor-kit/program-v2
Commit: cd09588d2459571ceb6fe7bd8764abb139b6d3de
Program ID: BiE9vSdz9MidUiyjVYsu3PG4C1fbPZ8CVPADA9jRfXw7
Audit Result:
• **Audited Commit:** 79cfc6c3fcd16df63011e507f8da09092c17dc40

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-C1	Lack of Inclusion of Accounts in Signed Payload	CRITICAL	FIXED
ACC-C2	RemoveAuthority Allows Cross-Wallet Authority Deletion	CRITICAL	FIXED
ACC-H1	Missing Accounts in Signed Payload Enables Unauthorized Authority Removal and Rent Theft	HIGH	FIXED
ACC-H2	Secp256r1 Authority Layout Mismatch Can Break Validation	HIGH	FIXED
ACC-M1	Old Nonces Can be Submitted Due To Truncation of Slot	MEDIUM	FIXED
ACC-M2	Missing Discriminator in Signed Payload Enables Signature Replay Across Instructions	MEDIUM	FIXED
ACC-L1	System program Account isn't checked.	LOW	FIXED
ACC-L2	Missing Payer in Signed Payload Enables Rent Extraction in Ownership Transfer	LOW	FIXED
ACC-L3	Secp256r1 Authenticator Allows Anyone to Submit Valid Signatures	LOW	FIXED
ACC-L4	Hardcoded rent calculations might go out of sync after chain update.	LOW	FIXED
ACC-L5	System Program Create Account Usage Leads to Lamport Transfer DoS	LOW	FIXED
ACC-I1	OOB Read On Get Slot Hash	INFO	FIXED
ACC-I2	Unintended Self-Reentrancy Risk	INFO	FIXED
ACC-I3	Wallet Validation Skips Discriminator Check	INFO	FIXED

DETAILED ISSUES

ACC-C1 Lack of Inclusion of Accounts in Signed Payload

CRITICAL FIXED

Description

We found that in the `execute` instruction, the signed payload only binds to the **index of accounts**, not the **exact account addresses**. Because of this, an attacker can reorder accounts OR submit any account in the transaction while still using a valid signature.

Example:

- User signs a payload intending:
- Transfer 1 token to **UserA**
- Transfer 100 tokens to **UserB**
- Accounts are signed by index: `[UserA, UserB]`
- An attacker submits the transaction with accounts reordered: `[UserB, UserA]`

As a result, the transfers execute with swapped recipients, causing unintended fund movement.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/execute.rs#L109

Relevant Code

Mitigation Suggestion

Include the **full account pubkeys** alongside the index, = in the signed payload instead of only account indices, so reordering accounts invalidates the signature.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/ffce84f249568de8b3fad8bbc644d599a537d76e.

ACC-C2 RemoveAuthority Allows Cross-Wallet Authority Deletion

CRITICAL FIXED

Description

We found that in In `manage_authority.rs:process_remove_authority`, when an **Owner** (role 0) removes an authority, the code does NOT verify that the target authority belongs to the same wallet. The entire authorization block is skipped when `admin_header.role == 0`:

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/manage_authority.rs#L377-L385

Relevant Code

```
// Authorization
if admin_header.role != 0 {
    // Only enters this block if NOT owner
    let target_data = unsafe { target_auth_pda.borrow_data_unchecked() };
    // ... reads target_header ...
    if target_header.discriminator != AccountDiscriminator::Authority as u8 {
        return Err(ProgramError::InvalidAccountData);
    }
    if admin_header.role != 1 || target_header.role != 2 {
        return Err(AuthError::PermissionDenied.into());
    }
}
```

Mitigation Suggestion

```
// ALWAYS read and verify target header
let target_data = unsafe { target_auth_pda.borrow_data_unchecked() };
let target_header = // ... parse header ...

// ALWAYS verify target belongs to this wallet
if target_header.wallet != *wallet_pda.key() {
    return Err(ProgramError::InvalidAccountData);
}
if target_header.discriminator != AccountDiscriminator::Authority as u8 {
    return Err(ProgramError::InvalidAccountData);
}

// Then check role-based permissions
if admin_header.role != 0 {
    if admin_header.role != 1 || target_header.role != 2 {
        return Err(AuthError::PermissionDenied.into());
    }
}
```

Remediation

No response

ACC-H1 Missing Accounts in Signed Payload Enables Unauthorized Authority Removal and Rent Theft

HIGH FIXED

Description

We found that `process_remove_authority` does not include `target_auth_pda` and `refund_dest` in the `signed_payload`. Because these accounts are not signed, a valid signature can be reused with different accounts.

As a result, an attacker can submit the same signature but replace:

- `target_auth_pda` with another user's authority PDA (to delete it), and
- `refund_dest` with their own account (to receive the reclaimed rent).

This allows unauthorized deletion of authority records and rent theft.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/manage_authority.rs#L299-L316

Relevant Code

```
let data_payload = &[]; // Empty for remove


    let account_info_iter = &mut accounts.iter();
    let _payer = account_info_iter
        .next()
        .ok_or(ProgramError::NotEnoughAccountKeys)?;
    let wallet_pda = account_info_iter
        .next()
        .ok_or(ProgramError::NotEnoughAccountKeys)?;
    let admin_auth_pda = account_info_iter
        .next()
        .ok_or(ProgramError::NotEnoughAccountKeys)?;
    let target_auth_pda = account_info_iter
        .next()
        .ok_or(ProgramError::NotEnoughAccountKeys)?;
    let refund_dest = account_info_iter
        .next()
        .ok_or(ProgramError::NotEnoughAccountKeys)?;
```

Mitigation Suggestion

Include `target_auth_pda` and `refund_dest` pubkeys in the `signed_payload` so the signature is bound to the exact accounts being removed and refunded.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/3c896dca89f3c4a79ab79db959efdf65b570dc33.

ACC-H2 Secp256r1 Authority Layout Mismatch Can Break Validation

HIGH FIXED

Description

We found an inconsistency in how Secp256r1 authority data is written vs read. When writing, the code inserts **four zero bytes** after the header, but when reading, those 4 bytes are not included in the offset calculation.

This makes the read logic interpret the wrong bytes as `rp_id_hash` (and later fields), which can cause failed validation or incorrect authority data parsing. The 4-byte prefix looks like a leftover or unfinished layout change (e.g., planned length/version field).

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/auth/secp256r1/mod.rs#L116

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_wallet.rs#L274-L275

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/transfer_ownership.rs#L234

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/manage_authority.rs#L265

Relevant Code

```
let stored_rp_id_hash = &auth_data[header_size..header_size + 32];


    if args.authority_type == 1 {
        variable_target[0..4].copy_from_slice(&0u32.to_le_bytes());
```

Mitigation Suggestion

Make read and write use the same fixed layout (either remove the extra 4 bytes or include them in the read offsets) and add a version/length field if the format is expected to evolve.

Remediation

Fixed by considering 4 zero bytes in `secp256r1/mod.rs#L79`.

ACC-M1 Old Nonces Can be Submitted Due To Truncation of Slot

MEDIUM FIXED

Description

We found that due to slot truncation, old hashes can be submitted. Slot 9050 will become valid in Slot 10050 again.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/auth/secp256r1/nonce.rs#L28-L52

Relevant Code

```
/// nonce.rs L28-L52
pub fn validate_nonce(
    slothashes_sysvar: &AccountInfo,
    submitted_slot: &TruncatedSlot,
) -> Result<[u8; 32], ProgramError> {
    // Ensure the program isn't being called via CPI
    if get_stack_height() > 1 {
        return Err(AuthError::PermissionDenied.into()); // Mapping CPINotAllowed error
    }

    let slothashes = SlotHashes::<Ref<[u8]>>::try_from(slothashes_sysvar)?;

    // Get current slothash (index 0)
    let most_recent_slot_hash = slothashes.get_slot_hash(0)?;
    let truncated_most_recent_slot = TruncatedSlot::new(most_recent_slot_hash.height);

    let index_difference = truncated_most_recent_slot.get_index_difference(submitted_slot);

    if index_difference >= 150 {
        return Err(AuthError::InvalidSignatureAge.into());
    }

    let slot_hash = slothashes.get_slot_hash(index_difference as usize)?;

    Ok(slot_hash.hash)
}
```

Mitigation Suggestion

Validate the full slot hash.

Remediation

Fixed in commit https://github.com/lazor-kit/program-v2/commit/30c9663f3da7c4386e50b7e797bc952c7890cdf4.

ACC-M2 Missing Discriminator in Signed Payload Enables Signature Replay Across Instructions

MEDIUM FIXED

Description

We found that the current implementation does not include the instruction discriminator (or any domain separator) in `signed_payload`. Because of this, a signature created for one instruction could potentially be used for a different instruction that builds the same payload format. This weakens authorization and can enable cross-instruction replay.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_session.rs#L150

Relevant Code

Mitigation Suggestion

Include the instruction discriminator (or a fixed domain separator string like `"create_session"`) in `signed_payload` so signatures are bound to a single instruction.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/5bf98cc10e8c7890528c96d3213411c9017a03e0.

ACC-L1 System program Account isn't checked.

LOW FIXED

Description

We found that the program isn't checking the system program account anywhere, allowing us to spoof it.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_wallet.rs#L230-L234

Relevant Code

```
/// create_wallet.rs L230-L234
    let create_auth_ix = Instruction {
        program_id: system_program.key(),
        accounts: &auth_accounts_meta,
        data: &create_auth_ix_data,
    };
```

Mitigation Suggestion

Check the system program id, or hardcode the Instruction program_id.

Remediation

Fixed in commit https://github.com/lazor-kit/program-v2/commit/cedb024349ecc737c980c64bb10171ac24a5e5d3.

ACC-L2 Missing Payer in Signed Payload Enables Rent Extraction in Ownership Transfer

LOW FIXED

Description

We found that `transfer_ownership` does not include the **payer** in the `signed_payload`, similar to issue #13. Because the payer is not bound by the signature, an attacker can replace it when submitting the transaction.

Attack scenario:

- The current owner is **auth type 1**.
- The new owner is **auth type 0**, which requires fewer lamports.
- The rent difference is refunded to the payer.
- An attacker supplies their own payer account and receives the refunded lamports.

This allows unauthorized rent extraction during ownership transfer.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/transfer_ownership.rs#L96-L98

Relevant Code

Mitigation Suggestion

Include the payer pubkey in the `signed_payload` so rent refunds are bound to the signer's intent.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/09fd896a52bcdf4ccb6be5b0711e112ae9b395a7.

ACC-L3 Secp256r1 Authenticator Allows Anyone to Submit Valid Signatures

LOW FIXED

Description

We found that `Secp256r1Authenticator` allows **anyone** to submit a transaction as long as they provide valid signature data. While this is not a security issue by itself, it weakens control if there is any mistake in the signed payload (for example, missing fields like in issue #8 or #11). In such cases, other users could reuse the same signature, leading to unintended execution.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/auth/secp256r1/mod.rs#L33

Relevant Code

Mitigation Suggestion

Bind the signature to a specific on-chain signer (e.g., require a signer account) so signatures cannot be reused by others. Signer could be an account that is checked to be the signer and included in the `signed_payload`.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/3c83deff43df435ccbc291b9c7174830a81380c1.

ACC-L4 Hardcoded rent calculations might go out of sync after chain update.

LOW FIXED

Description

We found that the multiple instruction hardcodes rent calculations instead of using the Rent `sysvar`

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_wallet.rs#L206-L210 https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_wallet.rs#L135-L141

Relevant Code

```
/// create_wallet.rs L135-L141
    // 897840 + (space * 6960)
    let rent_base = 897840u64;
    let rent_per_byte = 6960u64;
    let wallet_rent = (wallet_space as u64)
        .checked_mul(rent_per_byte)
        .and_then(|val| val.checked_add(rent_base))
        .ok_or(ProgramError::ArithmeticOverflow)?;
/// create_wallet.rs L206-L210
    // Rent calculation: 897840 + (space * 6960)
    let auth_rent = (auth_space as u64)
        .checked_mul(6960)
        .and_then(|val| val.checked_add(897840))
        .ok_or(ProgramError::ArithmeticOverflow)?;
```

Mitigation Suggestion

Use the Rent sysvar to calculate the correct amount of rent

Remediation

Fixed in commit https://github.com/lazor-kit/program-v2/commit/50bb5b7459af5091c3159dcc68a14f641272d394

ACC-L5 System Program Create Account Usage Leads to Lamport Transfer DoS

LOW FIXED

Description

We found that the program calls the System program's `create_account` instruction to initialize new accounts without checking the account's exising lamports. The System program's `create_account` instruction will fail and return an error when it tries to create an account which already contains any amount of lamports, which is a problem because anyone may transfer a small amount of lamports to the account to be created, effectively preventing the creation using `create_account`.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_wallet.rs#L143-L179 https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/manage_authority.rs#L206-L247

Relevant Code

```
/// create_wallet.rs L143-L179
    let mut create_wallet_ix_data = Vec::with_capacity(52);
    create_wallet_ix_data.extend_from_slice(&0u32.to_le_bytes());
    create_wallet_ix_data.extend_from_slice(&wallet_rent.to_le_bytes());
    create_wallet_ix_data.extend_from_slice(&(wallet_space as u64).to_le_bytes());
    create_wallet_ix_data.extend_from_slice(program_id.as_ref());

    let wallet_accounts_meta = [
        AccountMeta {
            pubkey: payer.key(),
            is_signer: true,
            is_writable: true,
        },
        AccountMeta {
            pubkey: wallet_pda.key(),
            is_signer: true, // Must be true even with invoke_signed
            is_writable: true,
        },
    ];
    let create_wallet_ix = Instruction {
        program_id: system_program.key(),
        accounts: &wallet_accounts_meta,
        data: &create_wallet_ix_data,
    };
    let wallet_bump_arr = [wallet_bump];
    let wallet_seeds = [
        Seed::from(b"wallet"),
        Seed::from(&args.user_seed),
        Seed::from(&wallet_bump_arr),
    ];
    let wallet_signer: Signer = (&wallet_seeds).into();

    invoke_signed(
        &create_wallet_ix,
        &[&payer.clone(), &wallet_pda.clone(), &system_program.clone()],
        &[wallet_signer],
    )?;
```

Mitigation Suggestion

To mitigate the issue, apply the manual transfer-allocate-assign pattern. First, transfer the required lamport amount to achieve rent exemption. This amount may be 0 if the account already has at least the amount of lamports required for the intended allocation size. Then, allocate the amount of bytes required and assign the account to the intended program.

Remediation

Fixed in commit https://github.com/lazor-kit/program-v2/commit/b36a4cca8f8da7ed7a2ee429b4d2f5d799a7131b.

ACC-I1 OOB Read On Get Slot Hash

INFO FIXED

Description

We found the `get_slot_hash` incorrectly validates `index`, potentially allowing an out of bounds read when `index == slothash.len`.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/auth/secp256r1/slothashes.rs#L76-L82

Relevant Code

```
/// slothashes.rs L76-L82
    #[inline(always)]
    pub fn get_slot_hash(&self, index: usize) -> Result<&SlotHash, ProgramError> {
        if index > self.get_slothashes_len() as usize {
            return Err(AuthError::PermissionDenied.into()); // Mapping generic error for simplicity
        }
        unsafe { Ok(self.get_slot_hash_unchecked(index)) }
    }
```

Mitigation Suggestion

Should be `if index >= self. get_slothashes_len() as usize {`

Remediation

Fixed in commit https://github.com/lazor-kit/program-v2/commit/5ba56b615b7bdbf572553d75446a71c94ec391e0

ACC-I2 Unintended Self-Reentrancy Risk

INFO FIXED

Description

Solana allows programs to invoke themselves via CPI (self-reentrancy), which may be risky if not explicitly accounted for. While the current utilization of a counter appears safe and unaffected, reentrancy may introduce unexpected behavior in future changes. Thus, it will be appropriate to proactively disable self-reentrancy unless it is an intentional design feature.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/execute.rs#L184

Relevant Code

Mitigation Suggestion

Disable re-entrancy from the CPIs.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/b87b0d1ad0fb3daef5e1d9e20a74145157a6b68b.

ACC-I3 Wallet Validation Skips Discriminator Check

INFO FIXED

Description

We found that wallet validation only checks the owner and does not verify the account discriminator. While this does not create an immediate issue, it allows other account types owned by the same program to potentially pass wallet checks, which is not intended.

Location

https://github.com/lazor-kit/program-v2/blob/cd09588d2459571ceb6fe7bd8764abb139b6d3de/program/src/processor/create_session.rs#L92-L94

Relevant Code

```
if wallet_pda.owner() != program_id || authorizer_pda.owner() != program_id {
        return Err(ProgramError::IllegalOwner);
    }
```

Mitigation Suggestion

Also validate the wallet account discriminator to ensure only real wallet accounts are accepted.

Remediation

Fixed in https://github.com/lazor-kit/program-v2/commit/ed95dfe01988224103daa32ee20f50454d168366 and https://github.com/lazor-kit/program-v2/commit/ed60b30.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.