QORTRACE · SMART CONTRACT AUDIT

Asteroid

4UeLCRqARmfb6e6KQijtiktqqXUxbfk6jZng7DhuBAGS

SECURITY	TRUST	FINDINGS
98	90	4
EXCELLENT	EXCELLENT	0C·0H·0M·2L·2I

TIER	STANDARD
LANGUAGE	RUST
SUBMISSION	GITHUB
MODEL	claude-sonnet-4-5-20250929
AUDIT ID	08df16f1-63a0-4cb8-93c9-c1946b096622

Executive summary

This Rust code defines the interface for a Solana SPL Token program, including error types and instruction serialization/deserialization. It handles standard token operations like minting, transferring, burning, and account management with multisig support. Overall, the code demonstrates defensive programming with comprehensive input validation and proper error handling for instruction parsing.

Findings by severity

Severity	Open	Resolved	Total
CRITICAL	0	0	0
HIGH	0	0	0
MEDIUM	0	0	0
LOW	2	0	2
INFO	2	0	2
TOTAL	**4**	**0**	**4**

Findings

LOW · 2

LOW Missing bounds check on multisig signer count in InitializeMultisig

`instruction.rs:unpack():~line 790`

The InitializeMultisig instruction unpacks 'm' directly from the byte stream without validating it falls within MIN_SIGNERS..=MAX_SIGNERS before use, although the builder functions do check. A malicious actor could craft raw instruction data bypassing these checks.

```
2 => {
    let &m = rest.first().ok_or(InvalidInstruction)?;
    Self::InitializeMultisig { m }
}
```

FIX: Add validation in the unpack path: after extracting 'm', verify '(MIN_SIGNERS..=MAX_SIGNERS).contains(&(m as usize))' before constructing InitializeMultisig. Similarly for InitializeMultisig2 at tag 19.

LOW Potential precision loss in UiAmountToAmount conversion

`instruction.rs:TokenInstruction:~line 390`

The UiAmountToAmount instruction accepts a string representing a decimal amount but the code only defines the instruction structure without implementing the parsing logic. If parsing is done elsewhere without proper bounds checking, it could lead to overflow or precision issues when converting large decimal strings to u64.

```
UiAmountToAmount {
    /// The `ui_amount` of tokens to reformat.
    ui_amount: &'a str,
}
```

FIX: Ensure the parsing implementation validates the ui_amount string for valid decimal format, checks for overflow when multiplying by 10^decimals, and handles edge cases like extremely large or small values before converting to u64.

INFO · 2

INFO AuthorityType does not derive Copy despite being a simple enum

`instruction.rs:~line 965`

AuthorityType is a simple 4-variant enum that could benefit from deriving Copy to avoid unnecessary cloning when passing by value. This is a minor code-quality improvement with no security impact.

```
#[repr(u8)]
#[derive(Clone, Debug, PartialEq)]
pub enum AuthorityType {
    MintTokens,
    FreezeAccount,
    AccountOwner,
    CloseAccount,
}
```

FIX: Add Copy to the derive list: #[derive(Clone, Copy, Debug, PartialEq)]. This allows the type to be passed by value more efficiently.

INFO Inconsistent error mapping in TokenError

`error.rs:try_from():~line 95`

TokenError implements TryFrom<u32> with a fallback to ProgramError::InvalidArgument for unknown error codes. While functional, this means an attacker could potentially trigger obscure error codes that map to InvalidArgument, making debugging harder. Not a security issue but could complicate error analysis.

```
_ => Err(ProgramError::InvalidArgument),
```

FIX: Consider logging unexpected error codes before returning InvalidArgument, or use a custom 'UnknownError' variant if the architecture permits, to aid in forensic analysis.

10-point trust check

Check	Result	Evidence
Access Control	PASS	All privileged operations (MintTo, Burn, SetAuthority, etc.) require explicit signer_pubkeys arrays and mark authority accounts as signers in AccountMeta construction.
Events Emitted	FAIL	No event emission logic is present in this interface code; it only defines instruction structures without runtime execution or logging.
Reentrancy Protection	PASS	This is an instruction interface layer with no direct state mutation or cross-contract calls; reentrancy concerns are handled by the runtime and program logic, not here.
Safe Math	PASS	Arithmetic operations use safe conversions (try_into, from_le_bytes) and the Overflow error variant exists to signal runtime arithmetic issues.
Input Validation	PASS	Unpack functions systematically check input lengths (split_first, get(..8), split_at(32)) and return InvalidInstruction on malformed data; builder functions validate signer counts.
No Hardcoded Secrets	PASS	No private keys, seeds, or secret values are hardcoded; all authority pubkeys are passed as parameters.
No Tx Origin For Auth	PASS	Authority is determined by explicit signer_pubkeys arrays and AccountMeta signer flags, not by transaction origin; Solana's model enforces explicit signatures.
Error Handling	PASS	All fallible operations return Result<_, ProgramError>, and the TokenError enum provides granular error variants with descriptive messages via ToStr implementation.
Documentation Present	PASS	Every instruction variant has detailed doc comments explaining expected accounts, behavior, and data layout; error variants also have doc strings.
Tests Referenced	PASS	A test module exists for error.rs (test_parse_error_from_primitive_exhaustive) ensuring error code round-tripping; no tests shown for instruction.rs but error handling is tested.