# Secure Coding Practices

Sebastian Fritsch

# # whoami

- Security Researcher
- Background in CTFs and Security Engineering
- At Neodyme since 2021
- 🐦 @siintemal

# # Who We Are

- 🇩🇪 Team of security researchers
- Found and reported 80+ vulnerabilities in Solana Core
- Found and reported critical bugs in many of the largest Solana DeFi protocols
- Here to make the Solana ecosystem more secure
- 🐦 @neodyme 🌐 neodyme.io

# The 80/20 of Solana Security

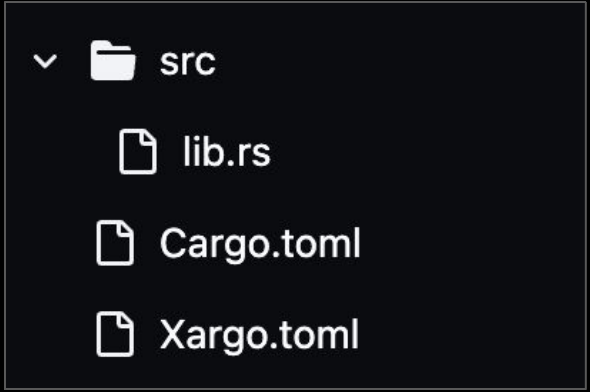
# # Triple S Framework

Structure

Safety

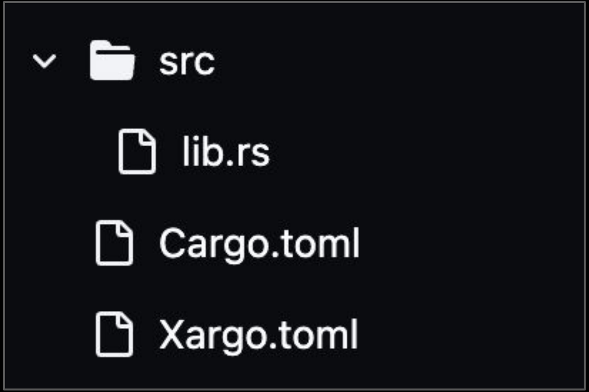
Supervision

# # Structure 📁

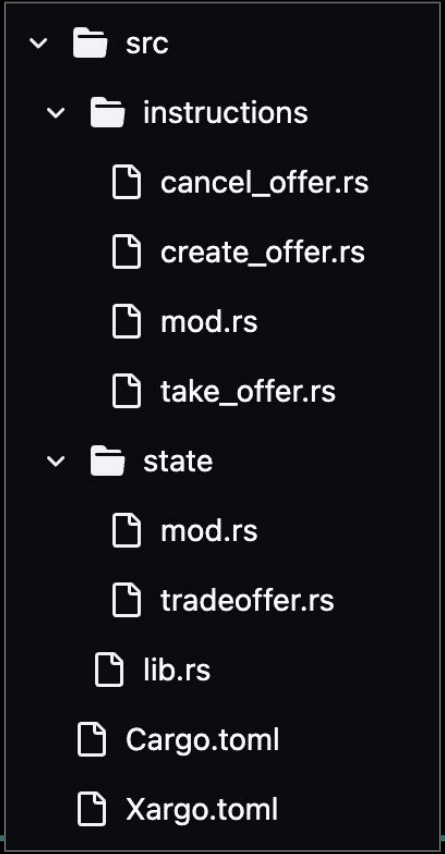


```
⌄ src
    lib.rs
  Cargo.toml
  Xargo.toml
```

# Structure 📁

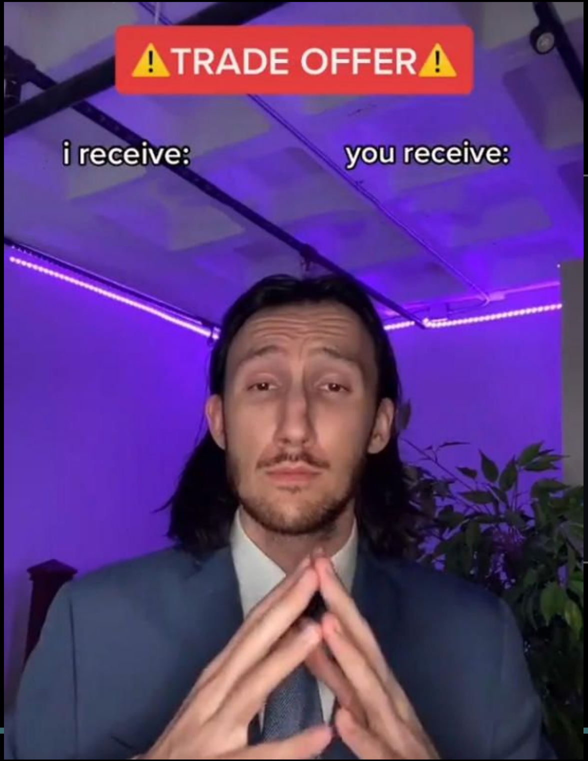


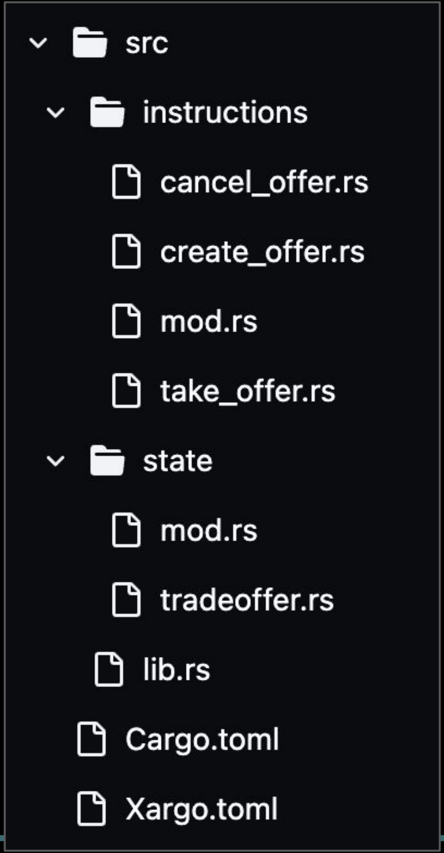
```
src
  lib.rs
Cargo.toml
Xargo.toml
```



```
src
  instructions
    cancel_offer.rs
    create_offer.rs
    mod.rs
    take_offer.rs
  state
    mod.rs
    tradeoffer.rs
  lib.rs
Cargo.toml
Xargo.toml
```

# Structure 📁

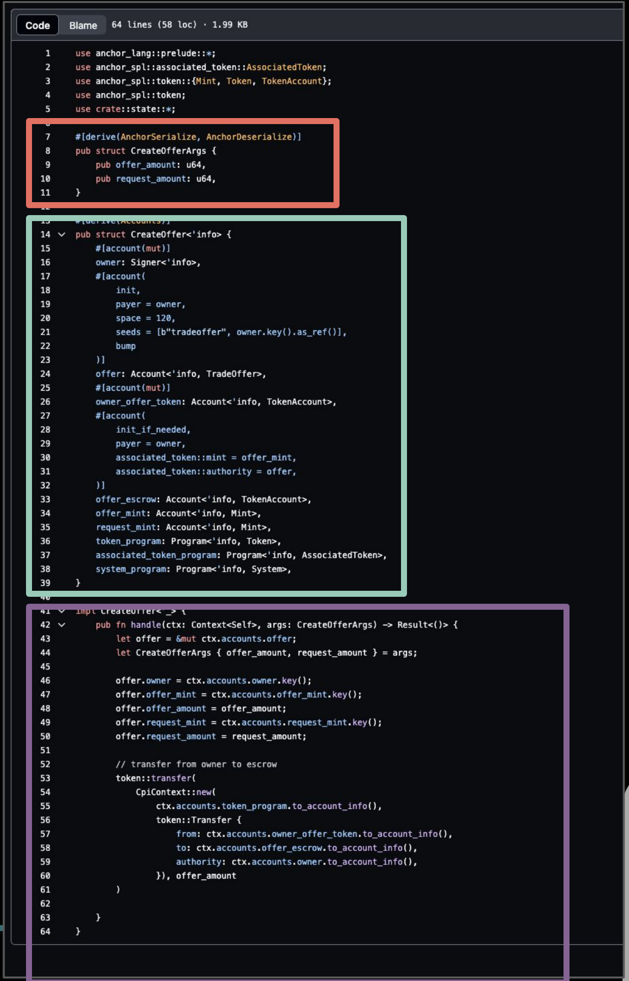




```
⌄ src
  ⌄ instructions
      cancel_offer.rs
      create_offer.rs
      mod.rs
      take_offer.rs
  ⌄ state
      mod.rs
      tradeoffer.rs
    lib.rs
  Cargo.toml
  Xargo.toml
```

# # Instruction File

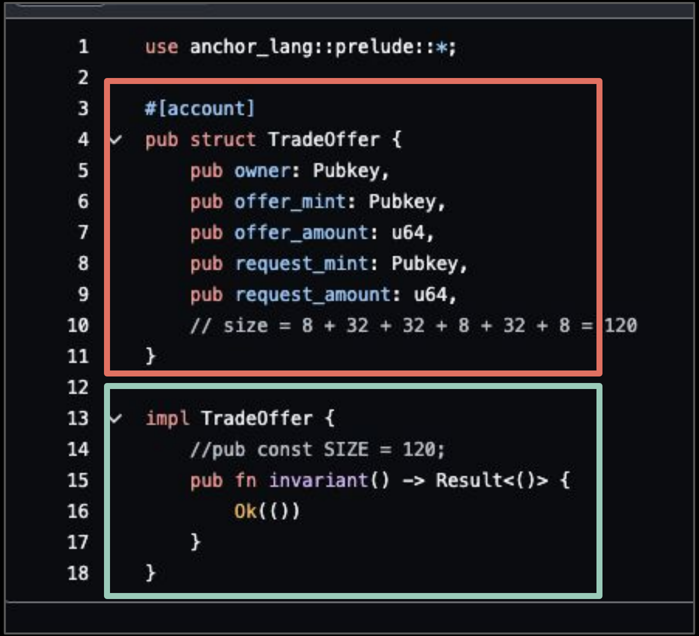
- Single file contains:
  - Instruction Argument Structure
  - Accounts Structure
  - Instruction Implementation



```rust
use anchor_lang::prelude::*;
use anchor_spl::associated_token::AssociatedToken;
use anchor_spl::token::{Mint, Token, TokenAccount};
use anchor_spl::token;
use crate::state::*;

#[derive(AnchorSerialize, AnchorDeserialize)]
pub struct CreateOfferArgs {
    pub offer_amount: u64,
    pub request_amount: u64,
}

#[derive(Accounts)]
pub struct CreateOffer<'info> {
    #[account(mut)]
    owner: Signer<'info>,
    #[account(
        init,
        payer = owner,
        space = 120,
        seeds = [b"tradeoffer", owner.key().as_ref()],
        bump
    )]
    offer: Account<'info, TradeOffer>,
    #[account(mut)]
    owner_offer_token: Account<'info, TokenAccount>,
    #[account(
        init_if_needed,
        payer = owner,
        associated_token::mint = offer_mint,
        associated_token::authority = offer,
    )]
    offer_escrow: Account<'info, TokenAccount>,
    offer_mint: Account<'info, Mint>,
    request_mint: Account<'info, Mint>,
    token_program: Program<'info, Token>,
    associated_token_program: Program<'info, AssociatedToken>,
    system_program: Program<'info, System>,
}

impl CreateOffer<'_> {
    pub fn handle(ctx: Context<Self>, args: CreateOfferArgs) -> Result<()> {
        let offer = &mut ctx.accounts.offer;
        let CreateOfferArgs { offer_amount, request_amount } = args;

        offer.owner = ctx.accounts.owner.key();
        offer.offer_mint = ctx.accounts.offer_mint.key();
        offer.offer_amount = offer_amount;
        offer.request_mint = ctx.accounts.request_mint.key();
        offer.request_amount = request_amount;

        // transfer from owner to escrow
        token::transfer(
            CpiContext::new(
                ctx.accounts.token_program.to_account_info(),
                token::Transfer {
                    from: ctx.accounts.owner_offer_token.to_account_info(),
                    to: ctx.accounts.offer_escrow.to_account_info(),
                    authority: ctx.accounts.owner.to_account_info(),
                }), offer_amount
        )

    }
}
```

# # State

- State Files contain:
    - Account struct definition
    - State implementations:
        - Verification logic
        - State transitions
        - Helper functions



```rust
use anchor_lang::prelude::*;

#[account]
pub struct TradeOffer {
    pub owner: Pubkey,
    pub offer_mint: Pubkey,
    pub offer_amount: u64,
    pub request_mint: Pubkey,
    pub request_amount: u64,
    // size = 8 + 32 + 32 + 8 + 32 + 8 = 120
}

impl TradeOffer {
    //pub const SIZE = 120;
    pub fn invariant() -> Result<()> {
        Ok(())
    }
}
```

# # Safety 🛡️

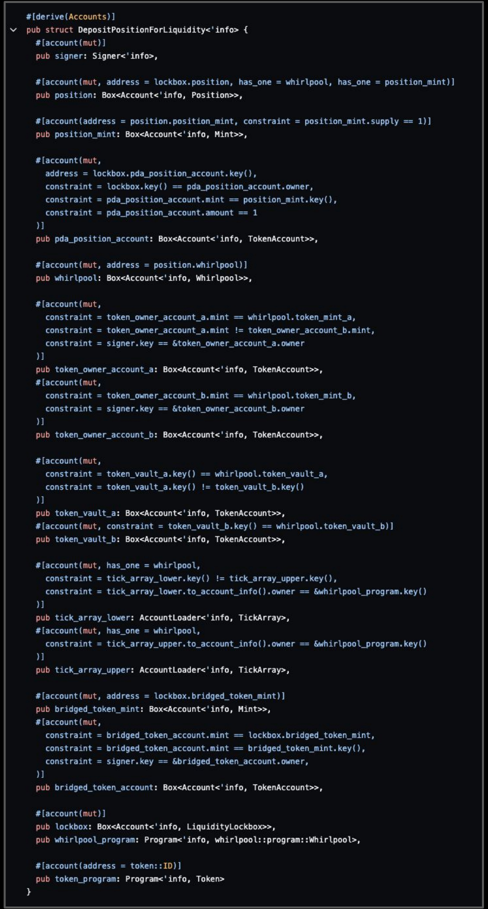
- Input validation
- Output validation
- Emergency mechanisms

# # Safety 🛡️

- Input validation
  - Constraints & Validation Function
- Output validation
  - Invariants & Assertions
- Emergency mechanisms
  - Program State & Circuit Breakers

# # Input Validation

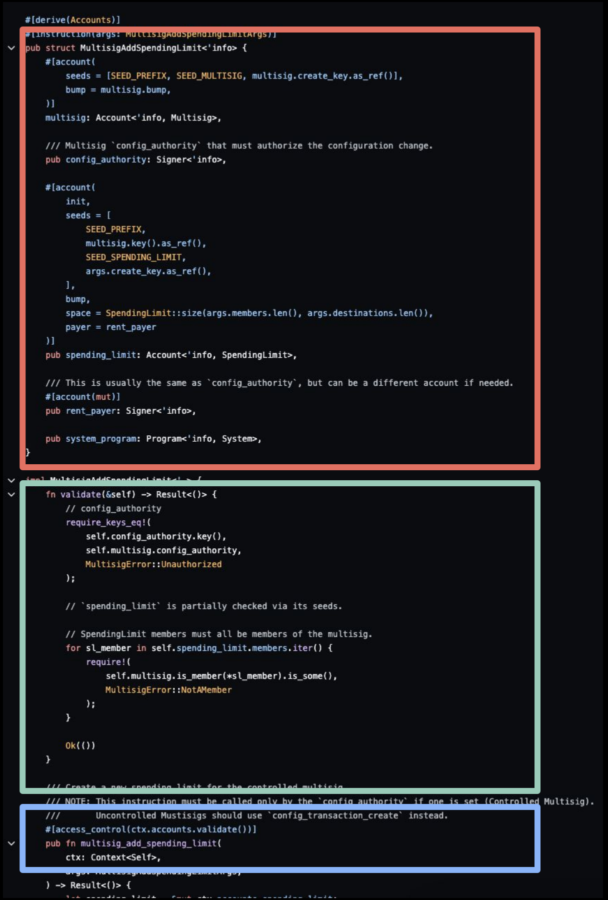
- Happens in the Instruction definition
- Use a mix of:
  1. Anchor constraints
  2. Separate Validation Function
- Constraints:
  - has_one, seeds
- Validation Function:
  - Everything else
- Screenshot tries to do everything in constraints 💀 →



```rust
#[derive(Accounts)]
pub struct DepositPositionForLiquidity<'info> {
  #[account(mut)]
  pub signer: Signer<'info>,

  #[account(mut, address = lockbox.position, has_one = whirlpool, has_one = position_mint)]
  pub position: Box<Account<'info, Position>>,

  #[account(address = position.position_mint, constraint = position_mint.supply == 1)]
  pub position_mint: Box<Account<'info, Mint>>,

  #[account(mut,
    address = lockbox.pda_position_account.key(),
    constraint = lockbox.key() == pda_position_account.owner,
    constraint = pda_position_account.mint == position_mint.key(),
    constraint = pda_position_account.amount == 1
  )]
  pub pda_position_account: Box<Account<'info, TokenAccount>>,

  #[account(mut, address = position.whirlpool)]
  pub whirlpool: Box<Account<'info, Whirlpool>>,

  #[account(mut,
    constraint = token_owner_account_a.mint == whirlpool.token_mint_a,
    constraint = token_owner_account_a.mint != token_owner_account_b.mint,
    constraint = signer.key == &token_owner_account_a.owner
  )]
  pub token_owner_account_a: Box<Account<'info, TokenAccount>>,
  #[account(mut,
    constraint = token_owner_account_b.mint == whirlpool.token_mint_b,
    constraint = signer.key == &token_owner_account_b.owner
  )]
  pub token_owner_account_b: Box<Account<'info, TokenAccount>>,

  #[account(mut,
    constraint = token_vault_a.key() == whirlpool.token_vault_a,
    constraint = token_vault_a.key() != token_vault_b.key()
  )]
  pub token_vault_a: Box<Account<'info, TokenAccount>>,
  #[account(mut, constraint = token_vault_b.key() == whirlpool.token_vault_b)]
  pub token_vault_b: Box<Account<'info, TokenAccount>>,

  #[account(mut, has_one = whirlpool,
    constraint = tick_array_lower.key() != tick_array_upper.key(),
    constraint = tick_array_lower.to_account_info().owner == &whirlpool_program.key()
  )]
  pub tick_array_lower: AccountLoader<'info, TickArray>,
  #[account(mut, has_one = whirlpool,
    constraint = tick_array_upper.to_account_info().owner == &whirlpool_program.key()
  )]
  pub tick_array_upper: AccountLoader<'info, TickArray>,

  #[account(mut, address = lockbox.bridged_token_mint)]
  pub bridged_token_mint: Box<Account<'info, Mint>>,
  #[account(mut,
    constraint = bridged_token_account.mint == lockbox.bridged_token_mint,
    constraint = bridged_token_account.mint == bridged_token_mint.key(),
    constraint = signer.key == &bridged_token_account.owner,
  )]
  pub bridged_token_account: Box<Account<'info, TokenAccount>>,

  #[account(mut)]
  pub lockbox: Box<Account<'info, LiquidityLockbox>>,
  pub whirlpool_program: Program<'info, whirlpool::program::Whirlpool>,

  #[account(address = token::ID)]
  pub token_program: Program<'info, Token>
}
```

# # Validation Function

- Seed checks done in constraints
- validate() function separate from business logic for additional checks
- Add the following before your ix handler function
  - #[access_control(ctx.accounts.validate())]



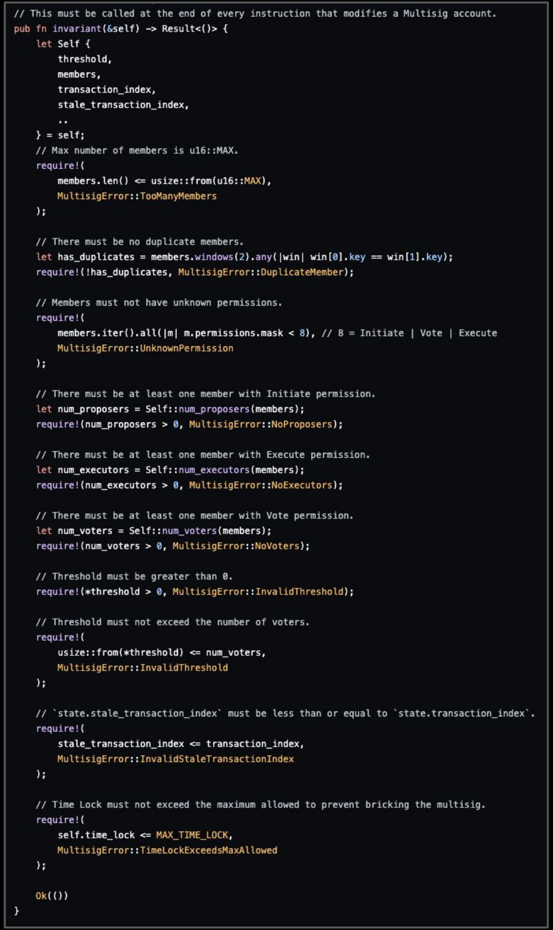
```rust
#[derive(Accounts)]
#[instruction(args: MultisigAddSpendingLimitArgs)]
pub struct MultisigAddSpendingLimit<'info> {
    #[account(
        seeds = [SEED_PREFIX, SEED_MULTISIG, multisig.create_key.as_ref()],
        bump = multisig.bump,
    )]
    multisig: Account<'info, Multisig>,

    /// Multisig `config_authority` that must authorize the configuration change.
    pub config_authority: Signer<'info>,

    #[account(
        init,
        seeds = [
            SEED_PREFIX,
            multisig.key().as_ref(),
            SEED_SPENDING_LIMIT,
            args.create_key.as_ref(),
        ],
        bump,
        space = SpendingLimit::size(args.members.len(), args.destinations.len()),
        payer = rent_payer
    )]
    pub spending_limit: Account<'info, SpendingLimit>,

    /// This is usually the same as `config_authority`, but can be a different account if needed.
    #[account(mut)]
    pub rent_payer: Signer<'info>,

    pub system_program: Program<'info, System>,
}

impl MultisigAddSpendingLimit<'_> {
    fn validate(&self) -> Result<()> {
        // config_authority
        require_keys_eq!(
            self.config_authority.key(),
            self.multisig.config_authority,
            MultisigError::Unauthorized
        );

        // `spending_limit` is partially checked via its seeds.

        // SpendingLimit members must all be members of the multisig.
        for sl_member in self.spending_limit.members.iter() {
            require!(
                self.multisig.is_member(*sl_member).is_some(),
                MultisigError::NotAMember
            );
        }

        Ok(())
    }

    /// Create a new spending limit for the controlled multisig.
    /// NOTE: This instruction must be called only by the `config_authority` if one is set (Controlled Multisig).
    ///       Uncontrolled Mustisigs should use `config_transaction_create` instead.
    #[access_control(ctx.accounts.validate())]
    pub fn multisig_add_spending_limit(
        ctx: Context<Self>,
        args: MultisigAddSpendingLimitArgs,
    ) -> Result<()> {
        let spending_limit = &mut ctx.accounts.spending_limit;
```

# # Output Validation

- Last step of an instruction fails → whole instruction fails
- So let’s validate our state at the very end!
    - Invariant functions
    - Assertions

# # Invariant Function

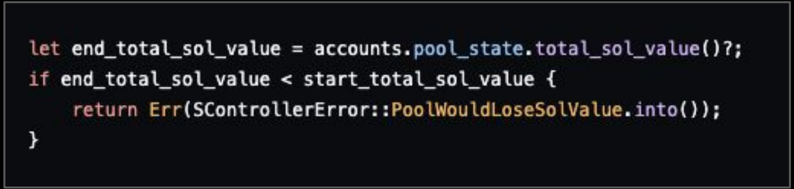
- Defined for some state account
- Called at the end of any instruction that changes this state
- Defines a list of requirements to the state
- If any requirement fails → rollback!
- Examples:
  - Multisig always has at least 1 member who can Execute proposals
  - Borrows never exceed deposits



```rust
// This must be called at the end of every instruction that modifies a Multisig account.
pub fn invariant(&self) -> Result<()> {
    let Self {
        threshold,
        members,
        transaction_index,
        stale_transaction_index,
        ..
    } = self;
    // Max number of members is u16::MAX.
    require!(
        members.len() <= usize::from(u16::MAX),
        MultisigError::TooManyMembers
    );

    // There must be no duplicate members.
    let has_duplicates = members.windows(2).any(|win| win[0].key == win[1].key);
    require!(!has_duplicates, MultisigError::DuplicateMember);

    // Members must not have unknown permissions.
    require!(
        members.iter().all(|m| m.permissions.mask < 8), // 8 = Initiate | Vote | Execute
        MultisigError::UnknownPermission
    );

    // There must be at least one member with Initiate permission.
    let num_proposers = Self::num_proposers(members);
    require!(num_proposers > 0, MultisigError::NoProposers);

    // There must be at least one member with Execute permission.
    let num_executors = Self::num_executors(members);
    require!(num_executors > 0, MultisigError::NoExecutors);

    // There must be at least one member with Vote permission.
    let num_voters = Self::num_voters(members);
    require!(num_voters > 0, MultisigError::NoVoters);

    // Threshold must be greater than 0.
    require!(*threshold > 0, MultisigError::InvalidThreshold);

    // Threshold must not exceed the number of voters.
    require!(
        usize::from(*threshold) <= num_voters,
        MultisigError::InvalidThreshold
    );

    // `state.stale_transaction_index` must be less than or equal to `state.transaction_index`.
    require!(
        stale_transaction_index <= transaction_index,
        MultisigError::InvalidStaleTransactionIndex
    );

    // Time Lock must not exceed the maximum allowed to prevent bricking the multisig.
    require!(
        self.time_lock <= MAX_TIME_LOCK,
        MultisigError::TimeLockExceedsMaxAllowed
    );

    Ok(())
}
```

# # Assertions

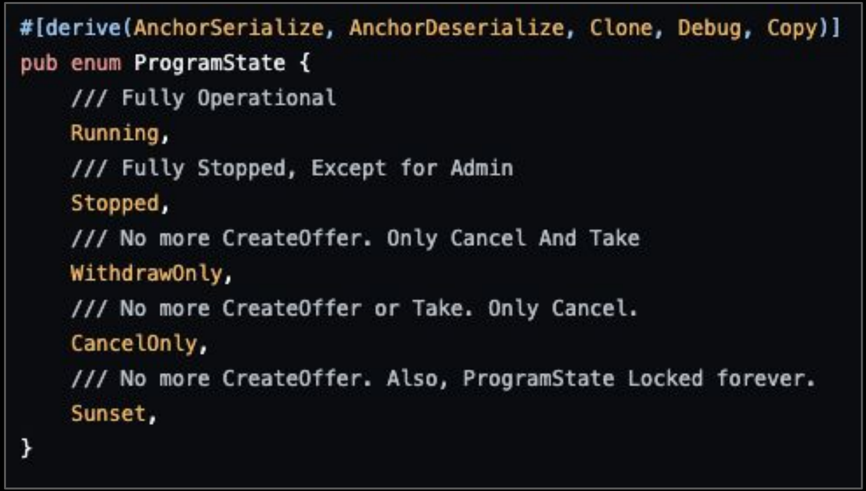
- Like invariants, but instruction-specific
- What effect would an exploit cause?
- Example:
  - Add Liquidity IX:
  - Check pool value at beginning and end
  - If value has decreased, fail!



```
let end_total_sol_value = accounts.pool_state.total_sol_value()?;
if end_total_sol_value < start_total_sol_value {
    return Err(SControllerError::PoolWouldLoseSolValue.into());
}
```

# # Emergency Mechanisms

- Have a global program state
  - Fully Operational
  - Fully Halted
  - Withdraw Only
  - Sunset
- Changeable by admin/multisig/dao…
- Consider Timelocks



```rust
#[derive(AnchorSerialize, AnchorDeserialize, Clone, Debug, Copy)]
pub enum ProgramState {
    /// Fully Operational
    Running,
    /// Fully Stopped, Except for Admin
    Stopped,
    /// No more CreateOffer. Only Cancel And Take
    WithdrawOnly,
    /// No more CreateOffer or Take. Only Cancel.
    CancelOnly,
    /// No more CreateOffer. Also, ProgramState Locked forever.
    Sunset,
}
```

# # Supervision 🔭

- Remember the Synthetify DAO hack?
  - Attacker prepared hack over multiple months
  - No one noticed …
- You need to keep track of your on-chain programs!
  - Monitor its state and usage
  - Monitor for anomalies
  - Detect attacks early

# # Logging

- Don’t use solana_program::log
    - CU expensive
    - Length limited
    - Hard to parse
    - Can be faked by other programs
- Use anchor’s event-cpi instead!
    - CU cheap
    - Verifiable
    - API ready

# # Monitoring

- Constantly running bot 🤖 that monitors your program
- Easily done with anchor events!
- Log events to discord/telegram
- Add sanity checks and alerts!
  - Large deposits/withdrawals
  - Abnormal behavior
  - Many hacks: Active user notices abnormal behaviour → hacks later from different account

# # Tests

1. Test for each instructions success case
2. Test for valid special cases
3. Test for each custom error (and have lots of custom errors!)

- Just testing for blueprint-execution, won't do anything for security

# # Summary

Structure 📁

Safety 🛡️

Supervision 🔭

# # Contact

Homepage: https://neodyme.io/

Blog:  https://blog.neodyme.io/

Twitter: @Neodyme

E-Mail: contact@neodyme.io

Me: sebastian@neodyme.io