Armors Labs

Fame Protocol

Smart Contract Audit

Fame Protocol Audit Summary

Project name : Fame Protocol Contract

Project address: None

Code URL : https://github.com/Fairmeme3/fame-program

Commit : b4c60b3376941b695052c4f439ff256cfac03ff3

Project target : Fame Protocol Contract Audit

Blockchain : Solana（SOL）

Test result : PASSED

Audit Info

Audit NO : 0X202404160006

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Fame Protocol Audit

The Fame Protocol team asked us to review and audit their Fame Protocol contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Fame Protocol Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2024-04-16

Audit results

Note that:

1. In theory, an overflow in multiply_and_divide function would return u64::MIN.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Fame Protocol contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing 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. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
./fame-program/src/lib.rs	2a2c27154b71d083de8cfd2f7e661715

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Integer Overflow and Underflow	safe
Replay	safe
Reordering	safe
Unsafe External Call	safe
Design Logic	safe
Scoping and Declarations	safe
Unsolved TODO comments	safe

Vulnerability	status
Forged account attack	safe
Race Conditions	safe
Denial Of Service (DOS)	safe
Arithmetic Accuracy Deviation	safe
Authority Control	safe

Contract file

```
use anchor_lang::prelude::*;
use anchor_lang::solana_program::native_token::LAMPORTS_PER_SOL;
use anchor_lang::solana_program::program::invoke_signed;
use anchor_lang::system_program::{transfer, Transfer};
use anchor_spl::associated_token::{get_associated_token_address, AssociatedToken};
use anchor_spl::metadata::mpl_token_metadata::types::DataV2;
use anchor_spl::metadata::{create_metadata_accounts_v3, CreateMetadataAccountsV3, Metadata};
use anchor_spl::token::{
    burn, mint_to, sync_native, transfer as spl_tranfer, Burn, Mint, MintTo, SyncNative, Token,
    TokenAccount, Transfer as SplTransfer,
};
use raydium_contract_instructions::amm_instruction;

declare_id!("6dHqbP7BTmvpyLaqfriXRz7Z6MC8PrkTFNu146hNUzBs");

// pub const START_TIMESTAMP: i64 = 1712476800;
// pub const END_TIMESTAMP: i64 = START_TIMESTAMP + 86400 * 2;
// pub const FINAL_REFUND_TIMESTAMP: i64 = END_TIMESTAMP + 86400;
// pub const LP_OPEN_TIME_DELAY: u64 = 3600;
// pub const MIN_FUNDS_THRESHOLD: u64 = 4000 * LAMPORTS_PER_SOL;
// pub const MAX_FUNDS_CAP: u64 = 5000 * LAMPORTS_PER_SOL;
// pub const MIN_FUNDS_PER_USER: u64 = LAMPORTS_PER_SOL / 10;

pub const START_TIMESTAMP: i64 = 1712556000;
pub const END_TIMESTAMP: i64 = START_TIMESTAMP + 30 * 60;
pub const FINAL_REFUND_TIMESTAMP: i64 = END_TIMESTAMP + 86400;
pub const LP_OPEN_TIME_DELAY: u64 = 3600;
pub const MIN_FUNDS_THRESHOLD: u64 = 20 * LAMPORTS_PER_SOL;
pub const MAX_FUNDS_CAP: u64 = 25 * LAMPORTS_PER_SOL;
pub const MIN_FUNDS_PER_USER: u64 = LAMPORTS_PER_SOL / 10;

// token details
pub const NAME: &str = "Fair Meme";
pub const SYMBOL: &str = "FAME";
pub const METADATA_URI: &str =
    "https://ipfs.io/ipfs/QmWGCS8ZeYVcjF6NhtycNBXHnCntBXWTXHMDW4xYMMHZfu";
pub const DECIMALS: u8 = 6;
pub const TOKEN_TOTAL_SUPPLY: u64 = 10u64.pow(9) * 10u64.pow(DECIMALS as u32);

fn multiply_and_divide(a: u64, b: u64, c: u64) -> u64 {
    // Try converting to u128
    if let (Ok(a_u128), Ok(b_u128), Ok(c_u128)) =
        (u128::try_from(a), u128::try_from(b), u128::try_from(c))
    {
        // Use u128 for calculation
        let result_u128: u128 = (a_u128 * b_u128) / c_u128;
        // Convert back to u64
        if let Ok(result_u64) = u64::try_from(result_u128) {
            return result_u64;
        }
    }
    // If conversion or calculation overflows, return u64's maximum value
    u64::MIN
}
#[program]
mod fame {
    use super::*;

    pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
        let vault = &mut ctx.accounts.vault;
        let fame_status = &mut ctx.accounts.fame_status;
        let signer = &mut ctx.accounts.signer;
        let system_program = &ctx.accounts.system_program;

        transfer(
            CpiContext::new(
                system_program.to_account_info(),
                Transfer {
                    from: signer.to_account_info(),
                    to: vault.to_account_info(),
                },
            ),
            3 * LAMPORTS_PER_SOL,
        )?;
        fame_status.current_funding_amount = 0;
        fame_status.max_funding_amount = 0;
        fame_status.funding_user_count = 0;
        fame_status.refunding_user_count = 0;
        fame_status.assigned_token_user_count = 0;
        fame_status.yield_to_hardcap_count = 0;
        fame_status.token_created = false;
        fame_status.lp_created = false;
        Ok(())
    }

    pub fn buy(ctx: Context<Buy>, amount: u64) -> Result<()> {
        let slot = Clock::get()?.unix_timestamp;
        require!(slot > START_TIMESTAMP, FameError::NotStartedYet);
        require!(slot < END_TIMESTAMP, FameError::HasEnded);
        require!(
            amount >= MIN_FUNDS_PER_USER,
            FameError::LessThanMinFundsPerUser
        );

        let vault = &mut ctx.accounts.vault;
        let fame_status = &mut ctx.accounts.fame_status;
        let signer = &mut ctx.accounts.signer;
        let system_program = &ctx.accounts.system_program;
        let vault_balance_before = vault.get_lamports();

        transfer(
            CpiContext::new(
                system_program.to_account_info(),
                Transfer {
                    from: signer.to_account_info(),
                    to: vault.to_account_info(),
                },
            ),
            amount,
        )?;

        let vault_balance_after = vault.get_lamports();

        require_eq!(vault_balance_after, vault_balance_before + amount);
        let current_share = &mut ctx.accounts.share;
        if current_share.funding_amount == 0 {
            fame_status.funding_user_count += 1;

            current_share.owner = signer.key();
            current_share.share_amount = 0;
            current_share.has_yield_to_hardcap = false;
            current_share.has_refunded = false;
            current_share.has_assigned_token = false;
        }
        current_share.funding_amount += amount;
        fame_status.current_funding_amount += amount;
        fame_status.max_funding_amount += amount;
        Ok(())
    }

    pub fn yield_to_hardcap(ctx: Context<Refund>) -> Result<()> {
        let slot = Clock::get()?.unix_timestamp;
        require!(slot > END_TIMESTAMP, FameError::NotEndedYet);
        let fame_status = &mut ctx.accounts.fame_status;
        require!(
            fame_status.max_funding_amount >= MIN_FUNDS_THRESHOLD,
            FameError::NotReachedMinFundsThreshold
        );
        let share = &mut ctx.accounts.share;
        let destination = &mut ctx.accounts.destination;

        require_eq!(share.owner, destination.key(), FameError::NotOwner);
        require!(
            !share.has_yield_to_hardcap,
            FameError::AlreadyYieldToHardcap
        );
        share.share_amount = multiply_and_divide(
            TOKEN_TOTAL_SUPPLY / 2,
            share.funding_amount,
            fame_status.max_funding_amount,
        );

        if fame_status.max_funding_amount > MAX_FUNDS_CAP {
            let yielded_funding_amount = multiply_and_divide(
                MAX_FUNDS_CAP,
                share.funding_amount,
                fame_status.max_funding_amount,
            );

            let refund_amount = share.funding_amount - yielded_funding_amount;

            let vault = &mut ctx.accounts.vault;
            let system_program = &ctx.accounts.system_program;

            let bump = &[ctx.bumps.vault];
            let seeds: &[&[u8]] = &[b"vault".as_ref(), bump];
            let signer_seeds = &[&seeds[..]];

            let vault_balance_before = vault.get_lamports();

            transfer(
                CpiContext::new(
                    system_program.to_account_info(),
                    Transfer {
                        from: vault.to_account_info(),
                        to: destination.to_account_info(),
                    },
                )
                .with_signer(signer_seeds),
                refund_amount,
            )?;
            let vault_balance_after = vault.get_lamports();
            require_eq!(vault_balance_after, vault_balance_before - refund_amount);

            share.funding_amount = yielded_funding_amount;
            fame_status.current_funding_amount -= refund_amount;
        }
        share.has_yield_to_hardcap = true;
        fame_status.yield_to_hardcap_count += 1;
        Ok(())
    }

    pub fn create_token(ctx: Context<CreateToken>) -> Result<()> {
        let fame_status = &mut ctx.accounts.fame_status;
        require!(
            fame_status.yield_to_hardcap_count == fame_status.funding_user_count,
            FameError::NotYieldToHardcapForAllUsers
        );
        require!(!fame_status.token_created, FameError::AlreadyCreatedToken);

        msg!("Running mint_to");
        let bump = &[ctx.bumps.mint];
        let seeds: &[&[u8]] = &[b"mint".as_ref(), bump];
        let signer_seeds = &[&seeds[..]];

        mint_to(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                MintTo {
                    authority: ctx.accounts.mint.to_account_info(),
                    to: ctx.accounts.vault_associated_token.to_account_info(),
                    mint: ctx.accounts.mint.to_account_info(),
                },
                signer_seeds,
            ),
            TOKEN_TOTAL_SUPPLY,
        )?;

        msg!("Run create metadata accounts v3");
        create_metadata_accounts_v3(
            CpiContext::new_with_signer(
                ctx.accounts.metadata_program.to_account_info(),
                CreateMetadataAccountsV3 {
                    payer: ctx.accounts.signer.to_account_info(),
                    mint: ctx.accounts.mint.to_account_info(),
                    metadata: ctx.accounts.metadata.to_account_info(),
                    mint_authority: ctx.accounts.mint.to_account_info(),
                    update_authority: ctx.accounts.mint.to_account_info(),
                    system_program: ctx.accounts.system_program.to_account_info(),
                    rent: ctx.accounts.rent.to_account_info(),
                },
                signer_seeds,
            ),
            DataV2 {
                name: String::from(NAME),
                symbol: String::from(SYMBOL),
                uri: String::from(METADATA_URI),
                seller_fee_basis_points: 0,
                creators: None,
                collection: None,
                uses: None,
            },
            true,
            true,
            None,
        )?;
        fame_status.token_created = true;
        Ok(())
    }

    pub fn assign_token(ctx: Context<AssignToken>) -> Result<()> {
        let fame_status = &mut ctx.accounts.fame_status;
        require!(fame_status.lp_created, FameError::LpNotCreatedYet);

        let share = &mut ctx.accounts.share;
        require!(!share.has_assigned_token, FameError::AlreadyAssignedToken);
        let destination = &mut ctx.accounts.destination;

        require_eq!(
            get_associated_token_address(&destination.key(), &ctx.accounts.mint.key()),
            ctx.accounts.to_associated_token.key(),
            FameError::IncorrectTokenVault
        );
        let (expected_share, _bump_seed) =
            Pubkey::find_program_address(&[destination.key().as_ref()], &ctx.program_id);

        require_eq!(share.owner, destination.key(), FameError::NotOwner);
        require_eq!(
            expected_share.key(),
            share.key(),
            FameError::IncorrectTokenVault
        );

        let bump = &[ctx.bumps.vault];
        let seeds: &[&[u8]] = &[b"vault".as_ref(), bump];
        let signer_seeds = &[&seeds[..]];

        spl_tranfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                SplTransfer {
                    from: ctx.accounts.vault_associated_token.to_account_info(),
                    to: ctx.accounts.to_associated_token.to_account_info(),
                    authority: ctx.accounts.vault.to_account_info(),
                },
                signer_seeds,
            ),
            share.share_amount,
        )?;
        share.has_assigned_token = true;
        fame_status.assigned_token_user_count += 1;
        Ok(())
    }

    pub fn refund(ctx: Context<Refund>) -> Result<()> {
        let slot = Clock::get()?.unix_timestamp;
        require!(slot > END_TIMESTAMP, FameError::NotEndedYet);
        let current_share = &mut ctx.accounts.share;
        require!(
            !current_share.has_refunded && current_share.funding_amount > 0,
            FameError::AlreadyRefunded
        );

        let vault = &mut ctx.accounts.vault;
        let fame_status = &mut ctx.accounts.fame_status;
        require!(
            fame_status.max_funding_amount < MIN_FUNDS_THRESHOLD
                || (slot > FINAL_REFUND_TIMESTAMP && !fame_status.lp_created),
            FameError::ReachedMinFundsThreshold
        );
        let destination = &mut ctx.accounts.destination;
        require_eq!(current_share.owner, destination.key(), FameError::NotOwner);

        let system_program = &ctx.accounts.system_program;

        let bump = &[ctx.bumps.vault];
        let seeds: &[&[u8]] = &[b"vault".as_ref(), bump];
        let signer_seeds = &[&seeds[..]];

        let vault_balance_before = vault.get_lamports();

        transfer(
            CpiContext::new_with_signer(
                system_program.to_account_info(),
                Transfer {
                    from: vault.to_account_info(),
                    to: destination.to_account_info(),
                },
                signer_seeds,
            ),
            current_share.funding_amount,
        )?;

        let vault_balance_after = vault.get_lamports();

        require_eq!(
            vault_balance_after,
            vault_balance_before - current_share.funding_amount
        );

        fame_status.refunding_user_count += 1;
        fame_status.current_funding_amount -= current_share.funding_amount;

        current_share.has_refunded = true;
        current_share.funding_amount = 0;

        Ok(())
    }

    pub fn initialize_lp(ctx: Context<InitializeLp>, nonce: u8) -> Result<()> {
        let fame_status = &mut ctx.accounts.fame_status;
        require!(
            fame_status.yield_to_hardcap_count == fame_status.funding_user_count,
            FameError::NotYieldToHardcapForAllUsers
        );
        require!(fame_status.token_created, FameError::TokenNotCreatedYet);
        require!(!fame_status.lp_created, FameError::AlreadyCreatedLp);

        let bump = &[ctx.bumps.vault];
        let seeds: &[&[u8]] = &[b"vault".as_ref(), bump];
        let signer_seeds = &[&seeds[..]];

        msg!("Running wrap sol to wsol");
        transfer(
            CpiContext::new_with_signer(
                ctx.accounts.system_program.to_account_info(),
                Transfer {
                    from: ctx.accounts.vault.to_account_info(),
                    to: ctx.accounts.user_pc_token_account.to_account_info(),
                },
                signer_seeds,
            ),
            fame_status.current_funding_amount,
        )?;

        // Sync the native token to reflect the new SOL balance as wSOL
        sync_native(CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            SyncNative {
                account: ctx.accounts.user_pc_token_account.to_account_info(),
            },
            signer_seeds,
        ))?;

        let opentime = Clock::get()?.unix_timestamp as u64 + LP_OPEN_TIME_DELAY;
        let coin_amount: u64 = TOKEN_TOTAL_SUPPLY / 2;
        let pc_amount: u64 = fame_status.current_funding_amount;

        msg!("Running raydium amm initialize2");
        let initialize_ix = amm_instruction::initialize2(
            ctx.accounts.amm_program.key,
            ctx.accounts.amm.key,
            ctx.accounts.amm_authority.key,
            ctx.accounts.amm_open_orders.key,
            ctx.accounts.lp_mint.key,
            ctx.accounts.coin_mint.key,
            ctx.accounts.pc_mint.key,
            ctx.accounts.pool_coin_token_account.key,
            ctx.accounts.pool_pc_token_account.key,
            ctx.accounts.amm_target_orders.key,
            ctx.accounts.amm_config.key,
            ctx.accounts.fee_destination.key,
            ctx.accounts.serum_program.key,
            ctx.accounts.serum_market.key,
            ctx.accounts.vault.key,
            ctx.accounts.user_coin_token_account.key,
            ctx.accounts.user_pc_token_account.key,
            &ctx.accounts.user_lp_token_account.key(),
            nonce,
            opentime,
            pc_amount,
            coin_amount,
        )?;
        let account_infos = [
            ctx.accounts.amm_program.clone(),
            ctx.accounts.amm.clone(),
            ctx.accounts.amm_authority.clone(),
            ctx.accounts.amm_open_orders.clone(),
            ctx.accounts.lp_mint.clone(),
            ctx.accounts.coin_mint.clone(),
            ctx.accounts.pc_mint.clone(),
            ctx.accounts.pool_coin_token_account.clone(),
            ctx.accounts.pool_pc_token_account.clone(),
            ctx.accounts.amm_target_orders.clone(),
            ctx.accounts.amm_config.clone(),
            ctx.accounts.fee_destination.clone(),
            ctx.accounts.serum_program.clone(),
            ctx.accounts.serum_market.clone(),
            ctx.accounts.vault.to_account_info().clone(),
            ctx.accounts.user_coin_token_account.clone(),
            ctx.accounts.user_pc_token_account.clone(),
            ctx.accounts.user_lp_token_account.clone(),
            ctx.accounts.token_program.to_account_info().clone(),
            ctx.accounts.system_program.to_account_info().clone(),
            ctx.accounts
                .associated_token_program
                .to_account_info()
                .clone(),
            ctx.accounts.rent.to_account_info().clone(),
        ];
        invoke_signed(&initialize_ix, &account_infos, signer_seeds)?;

        fame_status.lp_created = true;
        Ok(())
    }

    pub fn burn_lp_token(ctx: Context<BurnLpToken>) -> Result<()> {
        msg!("Burning raydium amm LP token");
        require!(
            ctx.accounts.fame_status.lp_created,
            FameError::LpNotCreatedYet
        );

        let bump = &[ctx.bumps.vault];
        let seeds: &[&[u8]] = &[b"vault".as_ref(), bump];
        let signer_seeds = &[&seeds[..]];
        burn(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                Burn {
                    from: ctx.accounts.user_lp_token_account.to_account_info(),
                    mint: ctx.accounts.lp_mint.to_account_info(),
                    authority: ctx.accounts.vault.to_account_info(),
                },
                signer_seeds,
            ),
            ctx.accounts.user_lp_token_account.amount,
        )?;
        Ok(())
    }
}

#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,

    #[account(init, seeds = [b"fame_status".as_ref()], bump, payer = signer, space = 8 + 34)]
    pub fame_status: Account<'info, FameStatus>,

    #[account(mut)]
    pub signer: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct Buy<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,

    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,

    #[account(init_if_needed, payer = signer, seeds = [signer.key().as_ref()], bump, space = 8 + 24 +
    pub share: Account<'info, Share>,
    #[account(mut)]
    pub signer: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct Refund<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,

    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,

    #[account(mut, seeds = [destination.key().as_ref()], bump)]
```

```rust
    pub share: Account<'info, Share>,
    /// CHECK: Safe
    #[account(mut)]
    pub destination: AccountInfo<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct CreateToken<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,
    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,
    #[account(mut)]
    pub signer: Signer<'info>,
    /// CHECK: Safe
    #[account(mut)]
    pub metadata: UncheckedAccount<'info>,
    #[account(
        init_if_needed,
        seeds = [b"mint".as_ref()],
        bump,
        payer = signer,
        mint::decimals = DECIMALS,
        mint::authority = mint,
    )]
    pub mint: Account<'info, Mint>,
    #[account(
        init_if_needed,
        payer = signer,
        associated_token::mint = mint,
        associated_token::authority = vault,
    )]
    pub vault_associated_token: Account<'info, TokenAccount>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub rent: Sysvar<'info, Rent>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    pub metadata_program: Program<'info, Metadata>,
}

#[derive(Accounts)]
pub struct AssignToken<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,
    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,
    #[account(mut)]
    pub signer: Signer<'info>,
    /// CHECK: Safe
    #[account(mut)]
    pub destination: AccountInfo<'info>,
    #[account(mut)]
    pub vault_associated_token: Account<'info, TokenAccount>,
    #[account(mut)]
    pub share: Account<'info, Share>,
    #[account(init_if_needed, payer = signer, associated_token::mint = mint, associated_token::author
    pub to_associated_token: Account<'info, TokenAccount>,
    #[account(mut, seeds = [b"mint".as_ref()], bump,)]
    pub mint: Account<'info, Mint>,
    pub token_program: Program<'info, Token>,
    pub system_program: Program<'info, System>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}
#[derive(Accounts)]
pub struct BurnLpToken<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,
    #[account(mut)]
    pub user_lp_token_account: Account<'info, TokenAccount>,
    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,
    /// CHECK: Safe
    pub serum_market: AccountInfo<'info>,
    /// CHECK: Safe
    pub amm_program: AccountInfo<'info>,
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"lp_mint_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    /// CHECK: Safe
    pub lp_mint: AccountInfo<'info>,
    pub token_program: Program<'info, Token>,
}
#[derive(Accounts)]
pub struct InitializeLp<'info> {
    #[account(mut, seeds = [b"vault".as_ref()], bump)]
    pub vault: SystemAccount<'info>,
    #[account(mut, seeds = [b"fame_status".as_ref()], bump)]
    pub fame_status: Account<'info, FameStatus>,
    /// CHECK: Safe
    pub amm_program: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"amm_associated_seed"],
        bump,
        seeds::program = amm_program.key
    )]
    pub amm: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        seeds = [b"amm_config_account_seed"],
        bump,
        seeds::program = amm_program.key
    )]
    pub amm_config: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        seeds = [b"amm authority"],
        bump,
        seeds::program = amm_program.key
    )]
    pub amm_authority: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"open_order_associated_seed"],
        bump,
        seeds::program = amm_program.key
    )]
    pub amm_open_orders: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"lp_mint_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    pub lp_mint: AccountInfo<'info>,
    /// CHECK: Safe
    pub coin_mint: AccountInfo<'info>,
    /// CHECK: Safe
    pub pc_mint: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"coin_vault_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    pub pool_coin_token_account: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"pc_vault_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    pub pool_pc_token_account: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"target_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    pub amm_target_orders: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(mut)]
    pub fee_destination: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(
        mut,
        seeds = [
            amm_program.key.as_ref(),
            serum_market.key.as_ref(),
            b"temp_lp_token_associated_seed"
        ],
        bump,
        seeds::program = amm_program.key
    )]
    pub pool_temp_lp: AccountInfo<'info>,
    /// CHECK: Safe
    pub serum_program: AccountInfo<'info>,
    /// CHECK: Safe
    pub serum_market: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(mut)]
    pub user_coin_token_account: AccountInfo<'info>,
    /// CHECK: Safe
    #[account(mut)]
    pub user_pc_token_account: AccountInfo<'info>,
    /// CHECK: Safe
    // #[account(mut)]
    #[account(mut)]
    pub user_lp_token_account: AccountInfo<'info>,

    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    pub system_program: Program<'info, System>,
    pub rent: Sysvar<'info, Rent>,
}

#[account]
pub struct FameStatus {
    current_funding_amount: u64,
    max_funding_amount: u64,
    funding_user_count: u32,
    refunding_user_count: u32,
    assigned_token_user_count: u32,
    yield_to_hardcap_count: u32,
    token_created: bool,
    lp_created: bool,
}
#[account]
pub struct Share {
    owner: Pubkey,
    funding_amount: u64,
    share_amount: u64,
    has_yield_to_hardcap: bool,
    has_refunded: bool,
    has_assigned_token: bool,
}

#[error_code]
pub enum FameError {
    #[msg("Not Started Yet")]
    NotStartedYet,
    #[msg("Has Ended")]
    HasEnded,
    #[msg("Not Ended Yet")]
    NotEndedYet,
    #[msg("Not Share Owner")]
    NotOwner,
    #[msg("Less Than Min Funds Per User")]
    LessThanMinFundsPerUser,
    #[msg("Already refunded")]
    AlreadyRefunded,
    #[msg("Missing holders")]
    MissingHolders,
    #[msg("Reached Min Funds Threshold")]
    ReachedMinFundsThreshold,
    #[msg("Not Reached Min Funds Threshold")]
    NotReachedMinFundsThreshold,
    #[msg("Already Yield To Hardcap")]
    AlreadyYieldToHardcap,
    #[msg("Not Yield To Hardcap For All Users")]
    NotYieldToHardcapForAllUsers,
    #[msg("Already Created Token")]
    AlreadyCreatedToken,
    #[msg("Token Not Created Yet")]
    TokenNotCreatedYet,
    #[msg("Already Created Lp")]
    AlreadyCreatedLp,
    #[msg("Lp Not Created Yet")]
    LpNotCreatedYet,
    #[msg("Already Assigned Token")]
    AlreadyAssignedToken,
    #[msg("Associated Token Address Mismatch")]
    AssociatedTokenAddressMismatch,
    #[msg("Incorrect Token Vault")]
    IncorrectTokenVault,
}
```

Analysis of audit results

Re-Entrancy

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Integer Overflow and Underflow

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Replay

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Reordering

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Unsafe External Call

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Design Logic

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Scoping and Declarations

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Forged account attack

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Accuracy Deviation

- **Detection results:**

```
PASSED!
```

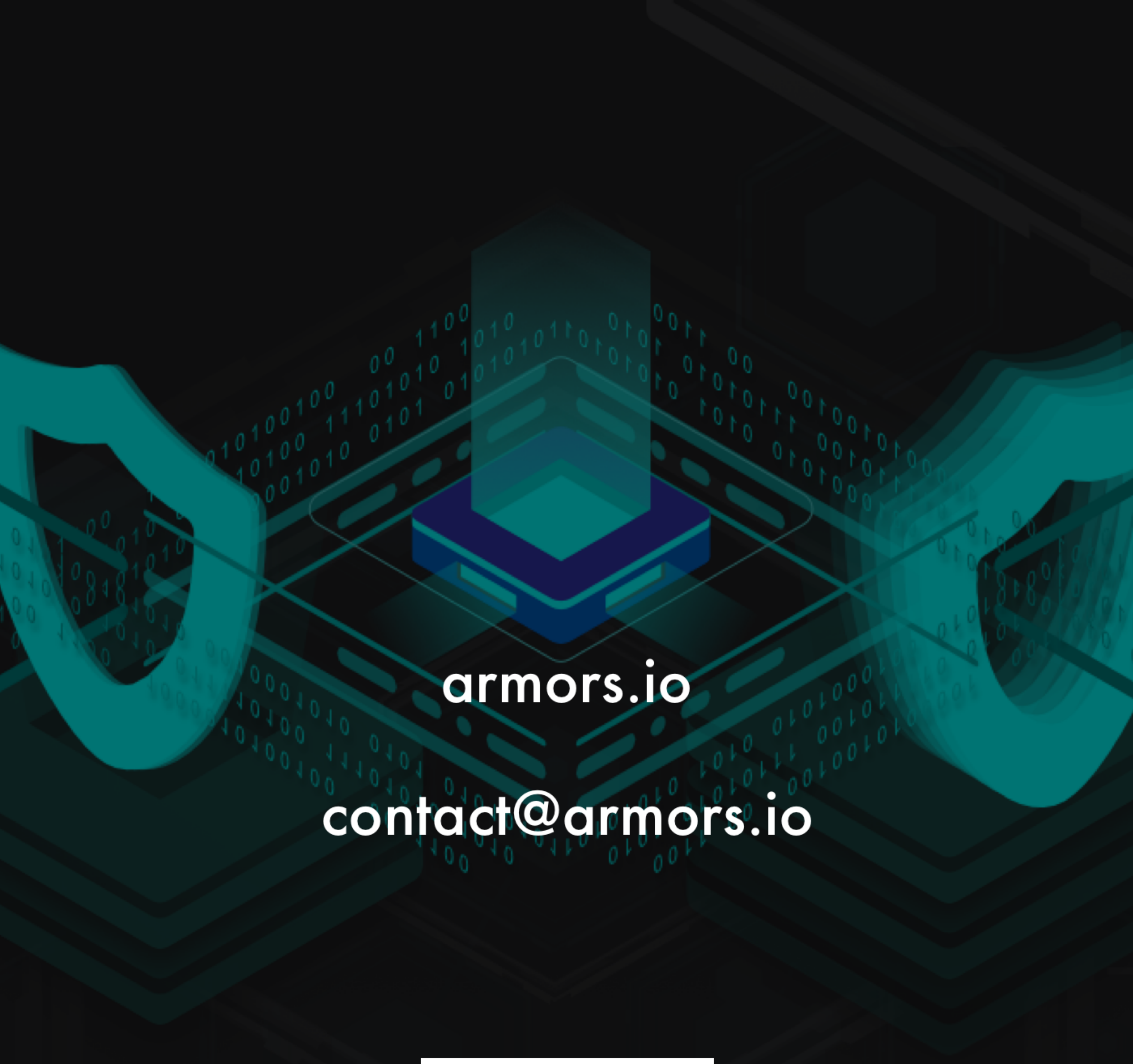
- **Security suggestion:**
  no.

Authority Control

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.



armors.io

contact@armors.io