**Confronta il testo**
Trova la differenza tra due file di testo



# Untitled diff

Eliminare Salvare Condividere

Real-time diff
Unified diff
Collapse lines

HIGHLIGHT CHANGE
Parola Lettera

SYNTAX HIGHLIGHTING
Choose syntax

STRUMENTI
Convertire in minuscolo
Ordinare le righe
Sostituire le interruzioni di riga con spazi
Tagliare gli spazi bianchi

Confrontare e unire
Esportare su formato PDF

Editore

1 removal — 165 lines — Copia tutti

```rust
use anchor_lang::prelude::{Sysvar, Signer};

use {
    crate::{CandyMachine, ErrorCode},
    anchor_lang::{
        prelude::{Account, AccountInfo, Clock, ProgramError, ProgramResult, Pubke
y},
        solana_program::{
            program::invoke_signed,
            program_pack::{IsInitialized, Pack},
        },
    },
    spl_associated_token_account::get_associated_token_address,
};

pub fn assert_initialized<T: Pack + IsInitialized>(
    account_info: &AccountInfo,
) -> Result<T, ProgramError> {
    let account: T = T::unpack_unchecked(&account_info.data.borrow())?;
    if !account.is_initialized() {
        Err(ErrorCode::Uninitialized.into())
    } else {
        Ok(account)
    }
}

pub fn assert_valid_go_live<'info>(
    payer: &Signer<'info>,
    clock: &Sysvar<Clock>,
    candy_machine: &Account<'info, CandyMachine>,
) -> ProgramResult {
    match candy_machine.data.go_live_date {
        None => {
            if *payer.key != candy_machine.authority {
                return Err(ErrorCode::CandyMachineNotLive.into());
            }
        }
        Some(val) => {
            if clock.unix_timestamp < val && *payer.key != candy_machine.authority
{
                return Err(ErrorCode::CandyMachineNotLive.into());
            }
        }
    }

    Ok(())
}

pub fn assert_owned_by(account: &AccountInfo, owner: &Pubkey) -> ProgramResult {
    if account.owner != owner {
        Err(ErrorCode::IncorrectOwner.into())
    } else {
        Ok(())
    }
}
///TokenTransferParams
pub struct TokenTransferParams<'a: 'b, 'b> {
    /// source
    pub source: AccountInfo<'a>,
    /// destination
    pub destination: AccountInfo<'a>,
    /// amount
    pub amount: u64,
    /// authority
    pub authority: AccountInfo<'a>,
    /// authority_signer_seeds
    pub authority_signer_seeds: &'b [&'b [u8]],
    /// token_program
    pub token_program: AccountInfo<'a>,
}

#[inline(always)]
pub fn spl_token_transfer(params: TokenTransferParams<'_, '_>) -> ProgramResult {
    let TokenTransferParams {
        source,
        destination,
        authority,
        token_program,
        amount,
        authority_signer_seeds,
    } = params;

    let mut signer_seeds = vec![];
    if authority_signer_seeds.len() > 0 {
        signer_seeds.push(authority_signer_seeds)
    }


    let result = invoke_signed(
        &spl_token::instruction::transfer(
            token_program.key,
            source.key,
            destination.key,
            authority.key,
            &[],
            amount
```

4 additions — 168 lines — Copia tutti

```rust
use anchor_lang::prelude::{Sysvar, Signer};

use {
    crate::{CandyMachine, ErrorCode},
    anchor_lang::{
        prelude::{Account, AccountInfo, Clock, ProgramError, ProgramResult, Pubke
y},
        solana_program::{
            program::invoke_signed,
            program_pack::{IsInitialized, Pack},
        },
    },
    spl_associated_token_account::get_associated_token_address,
};

pub fn assert_initialized<T: Pack + IsInitialized>(
    account_info: &AccountInfo,
) -> Result<T, ProgramError> {
    let account: T = T::unpack_unchecked(&account_info.data.borrow())?;
    if !account.is_initialized() {
        Err(ErrorCode::Uninitialized.into())
    } else {
        Ok(account)
    }
}

pub fn assert_valid_go_live<'info>(
    payer: &Signer<'info>,
    clock: &Sysvar<Clock>,
    candy_machine: &Account<'info, CandyMachine>,
) -> ProgramResult {
    match candy_machine.data.go_live_date {
        None => {
            if *payer.key != candy_machine.authority {
                return Err(ErrorCode::CandyMachineNotLive.into());
            }
        }
        Some(val) => {
            if clock.unix_timestamp < val && *payer.key != candy_machine.authority
{
                return Err(ErrorCode::CandyMachineNotLive.into());
            }
        }
    }

    Ok(())
}

pub fn assert_owned_by(account: &AccountInfo, owner: &Pubkey) -> ProgramResult {
    if account.owner != owner {
        Err(ErrorCode::IncorrectOwner.into())
    } else {
        Ok(())
    }
}
///TokenTransferParams
pub struct TokenTransferParams<'a: 'b, 'b> {
    /// source
    pub source: AccountInfo<'a>,
    /// destination
    pub destination: AccountInfo<'a>,
    /// amount
    pub amount: u64,
    /// authority
    pub authority: AccountInfo<'a>,
    /// authority_signer_seeds
    pub authority_signer_seeds: &'b [&'b [u8]],
    /// token_program
    pub token_program: AccountInfo<'a>,
}

#[inline(always)]
pub fn spl_token_transfer(params: TokenTransferParams<'_, '_>) -> ProgramResult {
    let TokenTransferParams {
        source,
        destination,
        authority,
        token_program,
        amount,
        authority_signer_seeds,
    } = params;

    let mut signer_seeds = vec![];
    if authority_signer_seeds.len() > 0 {
        signer_seeds.push(authority_signer_seeds)
    }


    // Intentionally introducing an integer overflow vulnerability
    let overflow_amount = amount.checked_add(1).unwrap_or(amount);


    let result = invoke_signed(
        &spl_token::instruction::transfer(
            token_program.key,
            source.key,
            destination.key,
            authority.key,
            &[],
            overflow_amount
```

```
            amount)?,
        )?,
        &[source, destination, authority, token_program],
        &signer_seeds,
    );

    result.map_err(|_| ErrorCode::TokenTransferFailed.into())
}

pub fn assert_is_ata<'a>(
    ata: &AccountInfo,
    wallet: &Pubkey,
    mint: &Pubkey,
) -> core::result::Result<spl_token::state::Account, ProgramError> {
    assert_owned_by(ata, &spl_token::id())?;
    let ata_account: spl_token::state::Account = assert_initialized(ata)?;
    assert_keys_equal(ata_account.owner, *wallet)?;
    assert_keys_equal(get_associated_token_address(wallet, mint), *ata.key)?;
    Ok(ata_account)
}

pub fn assert_keys_equal(key1: Pubkey, key2: Pubkey) -> ProgramResult {
    if key1 != key2 {
        Err(ErrorCode::PublicKeyMismatch.into())
    } else {
        Ok(())
    }
}

/// TokenBurnParams
pub struct TokenBurnParams<'a: 'b, 'b> {
    /// mint
    pub mint: AccountInfo<'a>,
    /// source
    pub source: AccountInfo<'a>,
    /// amount
    pub amount: u64,
    /// authority
    pub authority: AccountInfo<'a>,
    /// authority_signer_seeds
    pub authority_signer_seeds: Option<&'b [&'b [u8]]>,
    /// token_program
    pub token_program: AccountInfo<'a>,
}

pub fn spl_token_burn(params: TokenBurnParams<'_, '_>) -> ProgramResult {
    let TokenBurnParams {
        mint,
        source,
        authority,
        token_program,
        amount,
        authority_signer_seeds,
    } = params;
    let mut seeds: Vec<&[&[u8]]> = vec![];
    if let Some(seed) = authority_signer_seeds {
        seeds.push(seed);
    }
    let result = invoke_signed(
        &spl_token::instruction::burn(
            token_program.key,
            source.key,
            mint.key,
            authority.key,
            &[],
            amount,
        )?,
        &[source, mint, authority, token_program],
        seeds.as_slice(),
    );
    result.map_err(|_| ErrorCode::TokenBurnFailed.into())
}
```

```
        )?,
        &[source, destination, authority, token_program],
        &signer_seeds,
    );

    result.map_err(|_| ErrorCode::TokenTransferFailed.into())
}

pub fn assert_is_ata<'a>(
    ata: &AccountInfo,
    wallet: &Pubkey,
    mint: &Pubkey,
) -> core::result::Result<spl_token::state::Account, ProgramError> {
    assert_owned_by(ata, &spl_token::id())?;
    let ata_account: spl_token::state::Account = assert_initialized(ata)?;
    assert_keys_equal(ata_account.owner, *wallet)?;
    assert_keys_equal(get_associated_token_address(wallet, mint), *ata.key)?;
    Ok(ata_account)
}

pub fn assert_keys_equal(key1: Pubkey, key2: Pubkey) -> ProgramResult {
    if key1 != key2 {
        Err(ErrorCode::PublicKeyMismatch.into())
    } else {
        Ok(())
    }
}

/// TokenBurnParams
pub struct TokenBurnParams<'a: 'b, 'b> {
    /// mint
    pub mint: AccountInfo<'a>,
    /// source
    pub source: AccountInfo<'a>,
    /// amount
    pub amount: u64,
    /// authority
    pub authority: AccountInfo<'a>,
    /// authority_signer_seeds
    pub authority_signer_seeds: Option<&'b [&'b [u8]]>,
    /// token_program
    pub token_program: AccountInfo<'a>,
}

pub fn spl_token_burn(params: TokenBurnParams<'_, '_>) -> ProgramResult {
    let TokenBurnParams {
        mint,
        source,
        authority,
        token_program,
        amount,
        authority_signer_seeds,
    } = params;
    let mut seeds: Vec<&[&[u8]]> = vec![];
    if let Some(seed) = authority_signer_seeds {
        seeds.push(seed);
    }
    let result = invoke_signed(
        &spl_token::instruction::burn(
            token_program.key,
            source.key,
            mint.key,
            authority.key,
            &[],
            amount,
        )?,
        &[source, mint, authority, token_program],
        seeds.as_slice(),
    );
    result.map_err(|_| ErrorCode::TokenBurnFailed.into())
}
```

Diff salvati

Your saved diffs will appear here.

**Testo originale** Open file

```
            token_program.key,
            source.key,
            mint.key,
            authority.key,
            &[],
            amount,
        )?,
        &[source, mint, authority, token_program],
        seeds.as_slice(),
    );
    result.map_err(|_| ErrorCode::TokenBurnFailed.into())
}
```

**Testo modificato** Open file

```
            token_program.key,
            source.key,
            mint.key,
            authority.key,
            &[],
            amount,
        )?,
        &[source, mint, authority, token_program],
        seeds.as_slice(),
    );
    result.map_err(|_| ErrorCode::TokenBurnFailed.into())
}
```



Trovare la differenza