

Confronta il testo
Trova la differenza tra
due file di testo



Diffchecker Desktop
The most secure way to run Diffchecker. Get the Diffchecker Desktop app:
your diffs never leave your computer!

[Get Desktop](#)


Your new development career awaits. Check out the latest listings.
ADS VIA CARBON

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

Esporre su formato

PDF

Editor

Untitled diff

- 1 removal

165 lines Copia tutti

```

1 use anchor_lang::prelude::{Sysvar, Signer};
2
3 use {
4     crate::{CandyMachine, ErrorCode},
5     anchor_lang::{
6         prelude::{Account, AccountInfo, Clock, ProgramError, ProgramResult, Pubke
y},
7         solana_program::{
8             program::invoke_signed,
9             program_pack::{IsInitialized, Pack},
10        },
11    },
12    spl_associated_token_account::get_associated_token_address,
13 };
14
15 pub fn assert_initialized<T: Pack + IsInitialized>(
16     account_info: &AccountInfo,
17 ) -> Result<T, ProgramError> {
18     let account: T = T::unpack_unchecked(&account_info.data.borrow())?;
19     if !account.is_initialized() {
20         Err(ErrorCode::Uninitialized.into())
21     } else {
22         Ok(account)
23     }
24 }
25
26 pub fn assert_valid_go_live<'info>(
27     payer: &Signer<'info>,
28     clock: &Sysvar<Clock>,
29     candy_machine: &Account<'info, CandyMachine>,
30 ) -> ProgramResult {
31     match candy_machine.data.go_live_date {
32         None => {
33             if *payer.key != candy_machine.authority {
34                 return Err(ErrorCode::CandyMachineNotLive.into());
35             }
36         }
37         Some(val) => {
38             if clock.unix_timestamp < val && *payer.key != candy_machine.authority
39                 return Err(ErrorCode::CandyMachineNotLive.into());
40         }
41     }
42 }
43
44 Ok(())
45 }
46
47 pub fn assert_owned_by(account: &AccountInfo, owner: &Pubkey) -> ProgramResult {
48     if account.owner != owner {
49         Err(ErrorCode::IncorrectOwner.into())
50     } else {
51         Ok(())
52     }
53 }
54 ///TokenTransferParams
55 pub struct TokenTransferParams<'a: 'b, 'b> {
56     /// source
57     pub source: AccountInfo<'a>,
58     /// destination
59     pub destination: AccountInfo<'a>,
60     /// amount
61     pub amount: u64,
62     /// authority
63     pub authority: AccountInfo<'a>,
64     /// authority_signer_seeds
65     pub authority_signer_seeds: &'b [&'b [u8]],
66     /// token_program
67     pub token_program: AccountInfo<'a>,
68 }
69
70 #[inline(always)]
71 pub fn spl_token_transfer(params: TokenTransferParams<_, '_>) -> ProgramResult {
72     let TokenTransferParams {
73         source,
74         destination,
75         authority,
76         token_program,
77         amount,
78         authority_signer_seeds,
79     } = params;
80
81     let mut signer_seeds = vec![];
82     if authority_signer_seeds.len() > 0 {
83         signer_seeds.push(authority_signer_seeds)
84     }
85
86     let result = invoke_signed(
87         &spl_token::instruction::transfer(
88             token_program.key,
89             source.key,
90             destination.key,
91             authority.key,
92             &[],
93         )
94     );
95 }
```

```

86     let result = invoke_signed(
87         &spl_token::instruction::transfer(
88             token_program.key,
89             source.key,
90             destination.key,
91             authority.key,
92             &[],
93         )
94     );
95 }
```

+ 4 additions

168 lines Copia tutti

```

1 use anchor_lang::prelude::{Sysvar, Signer};
2
3 use {
4     crate::{CandyMachine, ErrorCode},
5     anchor_lang::{
6         prelude::{Account, AccountInfo, Clock, ProgramError, ProgramResult, Pubke
y},
7         solana_program::{
8             program::invoke_signed,
9             program_pack::{IsInitialized, Pack},
10        },
11    },
12    spl_associated_token_account::get_associated_token_address,
13 };
14
15 pub fn assert_initialized<T: Pack + IsInitialized>(
16     account_info: &AccountInfo,
17 ) -> Result<T, ProgramError> {
18     let account: T = T::unpack_unchecked(&account_info.data.borrow())?;
19     if !account.is_initialized() {
20         Err(ErrorCode::Uninitialized.into())
21     } else {
22         Ok(account)
23     }
24 }
25
26 pub fn assert_valid_go_live<'info>(
27     payer: &Signer<'info>,
28     clock: &Sysvar<Clock>,
29     candy_machine: &Account<'info, CandyMachine>,
30 ) -> ProgramResult {
31     match candy_machine.data.go_live_date {
32         None => {
33             if *payer.key != candy_machine.authority {
34                 return Err(ErrorCode::CandyMachineNotLive.into());
35             }
36         }
37         Some(val) => {
38             if clock.unix_timestamp < val && *payer.key != candy_machine.authority
39                 return Err(ErrorCode::CandyMachineNotLive.into());
40         }
41     }
42 }
43
44 Ok(())
45 }
46
47 pub fn assert_owned_by(account: &AccountInfo, owner: &Pubkey) -> ProgramResult {
48     if account.owner != owner {
49         Err(ErrorCode::IncorrectOwner.into())
50     } else {
51         Ok(())
52     }
53 }
54 ///TokenTransferParams
55 pub struct TokenTransferParams<'a: 'b, 'b> {
56     /// source
57     pub source: AccountInfo<'a>,
58     /// destination
59     pub destination: AccountInfo<'a>,
60     /// amount
61     pub amount: u64,
62     /// authority
63     pub authority: AccountInfo<'a>,
64     /// authority_signer_seeds
65     pub authority_signer_seeds: &'b [&'b [u8]],
66     /// token_program
67     pub token_program: AccountInfo<'a>,
68 }
69
70 #[inline(always)]
71 pub fn spl_token_transfer(params: TokenTransferParams<_, '_>) -> ProgramResult {
72     let TokenTransferParams {
73         source,
74         destination,
75         authority,
76         token_program,
77         amount,
78         authority_signer_seeds,
79     } = params;
80
81     let mut signer_seeds = vec![];
82     if authority_signer_seeds.len() > 0 {
83         signer_seeds.push(authority_signer_seeds)
84     }
85
86     // Intentionally introducing an integer overflow vulnerability
87     let overflow_amount = amount.checked_add(1).unwrap_or(amount);
88
89     let result = invoke_signed(
90         &spl_token::instruction::transfer(
91             token_program.key,
92             source.key,
93             destination.key,
94             authority.key,
95             &[],
96         )
97     );
98 }
```

overflow_amount

```

94     )?,
95     &[source, destination, authority, token_program],
96     &signer_seeds,
97   );
98
99   result.map_err(|_| ErrorCode::TokenTransferFailed.into())
100 }
101
102 pub fn assert_is_at<'a>(
103   ata: &AccountInfo,
104   wallet: &Pubkey,
105   mint: &Pubkey,
106 ) -> core::result::Result<spl_token::state::Account, ProgramError> {
107   assert_owed_by(ata, &spl_token::id())?;
108   let ata_account: spl_token::state::Account = assert_initialized(ata)?;
109   assert_keys_equal(ata_account.owner, *wallet)?;
110   assert_keys_equal(get_associated_token_address(wallet, mint), *ata.key)?;
111   Ok(ata_account)
112 }
113
114 pub fn assert_keys_equal(key1: Pubkey, key2: Pubkey) -> ProgramResult {
115   if key1 != key2 {
116     Err(ErrorCode::PublicKeyMismatch.into())
117   } else {
118     Ok(())
119   }
120 }
121
122 /// TokenBurnParams
123 pub struct TokenBurnParams<'a: 'b, 'b> {
124   /// mint
125   pub mint: AccountInfo<'a>,
126   /// source
127   pub source: AccountInfo<'a>,
128   /// amount
129   pub amount: u64,
130   /// authority
131   pub authority: AccountInfo<'a>,
132   /// authority_signer_seeds
133   pub authority_signer_seeds: Option<&'b [&'b [u8]]>,
134   /// token_program
135   pub token_program: AccountInfo<'a>,
136 }
137
138 pub fn spl_token_burn(params: TokenBurnParams<_, _>) -> ProgramResult {
139   let TokenBurnParams {
140     mint,
141     source,
142     authority,
143     token_program,
144     amount,
145     authority_signer_seeds,
146   } = params;
147   let mut seeds: Vec<&[u8]> = vec![];
148   if let Some(seed) = authority_signer_seeds {
149     seeds.push(seed);
150   }
151   let result = invoke_signed(
152     &spl_token::instruction::burn(
153       token_program.key,
154       source.key,
155       mint.key,
156       authority.key,
157       &[],
158       amount,
159     )?,
160     &[source, mint, authority, token_program],
161     seeds.as_slice(),
162   );
163   result.map_err(|_| ErrorCode::TokenBurnFailed.into())
164 }
165

```

```

97     )?,
98     &[source, destination, authority, token_program],
99     &signer_seeds,
100   );
101
102   result.map_err(|_| ErrorCode::TokenTransferFailed.into())
103 }
104
105 pub fn assert_is_at<'a>(
106   ata: &AccountInfo,
107   wallet: &Pubkey,
108   mint: &Pubkey,
109 ) -> core::result::Result<spl_token::state::Account, ProgramError> {
110   assert_owed_by(ata, &spl_token::id())?;
111   let ata_account: spl_token::state::Account = assert_initialized(ata)?;
112   assert_keys_equal(ata_account.owner, *wallet)?;
113   assert_keys_equal(get_associated_token_address(wallet, mint), *ata.key)?;
114   Ok(ata_account)
115 }
116
117 pub fn assert_keys_equal(key1: Pubkey, key2: Pubkey) -> ProgramResult {
118   if key1 != key2 {
119     Err(ErrorCode::PublicKeyMismatch.into())
120   } else {
121     Ok(())
122   }
123 }
124
125 /// TokenBurnParams
126 pub struct TokenBurnParams<'a: 'b, 'b> {
127   /// mint
128   pub mint: AccountInfo<'a>,
129   /// source
130   pub source: AccountInfo<'a>,
131   /// amount
132   pub amount: u64,
133   /// authority
134   pub authority: AccountInfo<'a>,
135   /// authority_signer_seeds
136   pub authority_signer_seeds: Option<&'b [&'b [u8]]>,
137   /// token_program
138   pub token_program: AccountInfo<'a>,
139 }
140
141 pub fn spl_token_burn(params: TokenBurnParams<_, _>) -> ProgramResult {
142   let TokenBurnParams {
143     mint,
144     source,
145     authority,
146     token_program,
147     amount,
148     authority_signer_seeds,
149   } = params;
150   let mut seeds: Vec<&[u8]> = vec![];
151   if let Some(seed) = authority_signer_seeds {
152     seeds.push(seed);
153   }
154   let result = invoke_signed(
155     &spl_token::instruction::burn(
156       token_program.key,
157       source.key,
158       mint.key,
159       authority.key,
160       &[],
161       amount,
162     )?,
163     &[source, mint, authority, token_program],
164     seeds.as_slice(),
165   );
166   result.map_err(|_| ErrorCode::TokenBurnFailed.into())
167 }
168

```

Diff salvati	Testo originale	Open file	Testo modificato	Open file
Your saved diffs will appear here.	<pre> 153 token_program.key, 154 source.key, 155 mint.key, 156 authority.key, 157 &[], 158 amount, 159)?, 160 &[source, mint, authority, token_program], 161 seeds.as_slice(), 162 }; 163 result.map_err(_ ErrorCode::TokenBurnFailed.into()) 164 } 165 </pre>		<pre> 156 token_program.key, 157 source.key, 158 mint.key, 159 authority.key, 160 &[], 161 amount, 162)?, 163 &[source, mint, authority, token_program], 164 seeds.as_slice(), 165 }; 166 result.map_err(_ ErrorCode::TokenBurnFailed.into()) 167 } 168 </pre>	



Bibcitation

A free online tool to generate citations, reference lists, and bibliographies. APA, MLA, Chicago, and more.

[Check it out](#)



SVG Viewer

View, optimize, convert, and share your SVGs easily on the web! Also the most advanced icon search

[Try it out](#)

[Trovare la differenza](#)