Solana Auditors Bootcamp

1. Intro & Advanced Anchor

ackee
blockchain security

Hi 👋

Andrej, @andrej_19

Auditor @ Ackee Blockchain Security

@AckeeBlockchain
ackee.xyz

ackee
blockchain security

Hi 👋

Josef Gattermayer, Ph.D.

CEO @ Ackee Blockchain Security

@AckeeBlockchain
ackee.xyz

ackee

blockchain security

About us

Blockchain security audits 🔎

Development of open source security tools 👩‍💻

Onboarding developers to blockchain 🎓

@AckeeBlockchain

ackee.xyz

ackee
blockchain security

We work with

1inch
LIDO
Safe
AXELAR
LayerZero.
Trader Joe
PENDLE
Marinade

Grants

ethereum foundation
coinbase
SOLANA FOUNDATION

ackee
blockchain security

What is this course about ?

ackee
blockchain security

The Course

- Advanced Anchor
- Integration Tests and Unit Tests
- Fuzzing with Trident I
- Fuzzing with Trident II
- Security Best Practices
- Common Vulnerability Vectors
- Capture the Flag

ackee
blockchain security

What this course is not about ?

- Introduction to Rust
- Introduction to Solana
- Introduction to Blockchain

ackee
blockchain security

Advanced Anchor Framework

ackee
blockchain security

Agenda

- How Anchor improves Vanilla Solana ?
- What is the #[program] ?
- What is the #[derive(Accounts)] ?
- What is the #[account] ?
- Why are the Account types important ?
- What are all the Account Constraints ?
- What is the Anchor IDL ?

ackee
blockchain security

What is Vanilla Solana ?

Vanilla Solana

vanilla-solana.rs

```rust
// declare and export the program's entrypoint
entrypoint!(process_instruction);

// program entrypoint's implementation
pub fn process_instruction(
    _program_id: &Pubkey,
    _accounts: &[AccountInfo],
    _instruction_data: &[u8],
) -> ProgramResult {
    // log a message to the blockchain
    msg!("Hello, world!");

    // gracefully exit the program
    Ok(())
}
```

Anchor

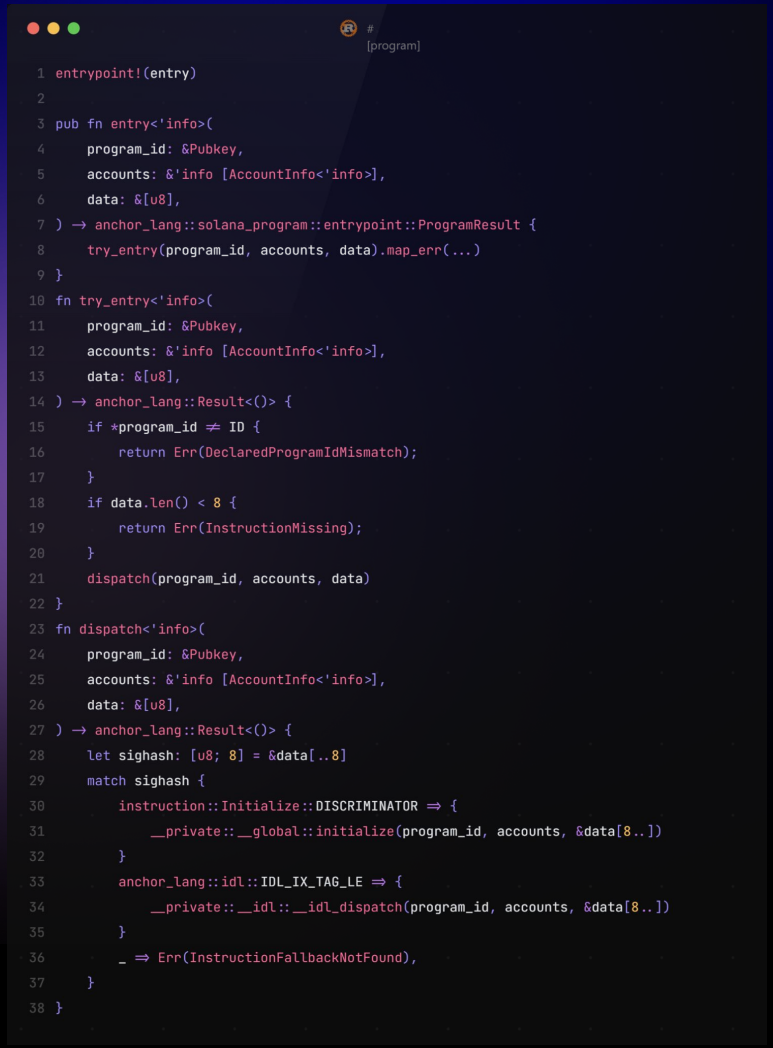
Anchor

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

declare_id!("HrxxFK82k8J88kUvDZvTZHnqCVbeyjvebwfUQHwET4B7");

#[program]
pub mod anchor_solana {
    use super::*;

    pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
        Ok(())
    }
}

#[derive(Accounts)]
pub struct Initialize {}
```

ackee
blockchain security

#[program]

- Encapsulates all the instruction **handlers that your program supports**.
- Encapsulates all the instructions **handlers for IDL manipulation**.
- Supports custom error types for **robust error management**.
- Works in conjunction with **#[derive(Accounts)]**.



ackee
blockchain security

#[derive(Accounts)]

- **Validate the accounts** passed to the instruction handlers.
- **Deserializes** Account data.
- Allows specifying various **constraints**, such as **#[account(init)]**.
- Generates **CPI struct** of the Accounts.

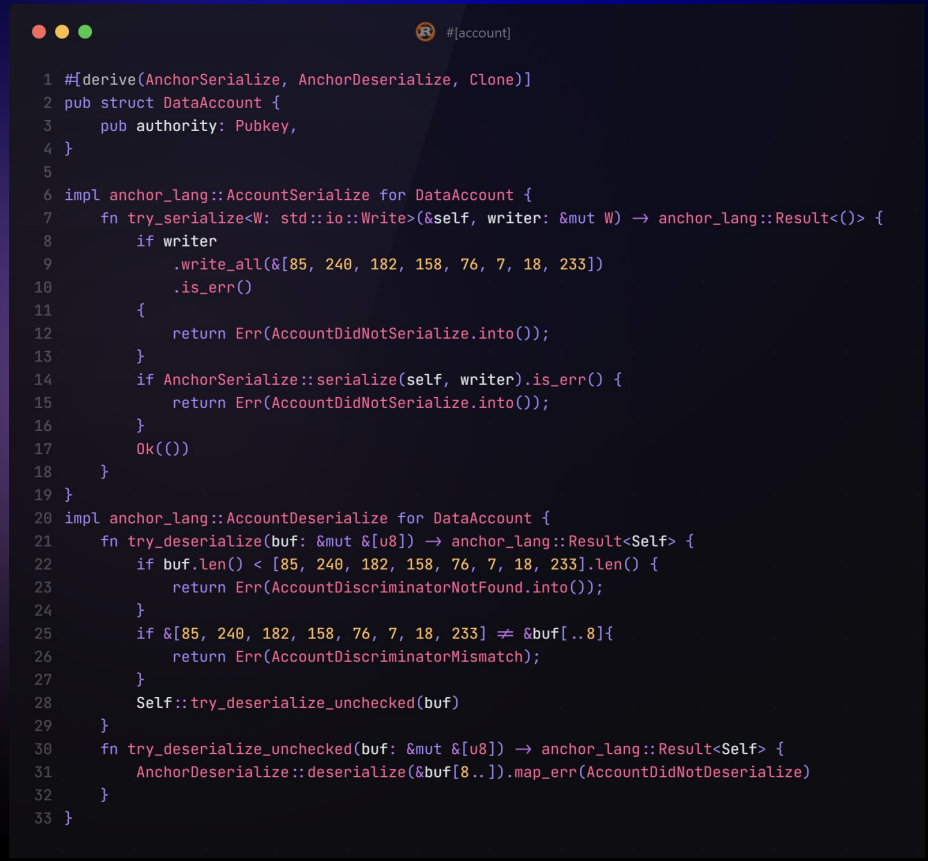
#[derive(Accounts)]

```rust
impl<'info> anchor_lang::Accounts<'info, InitializeBumps> for Initialize<'info>
where
    'info: 'info,
{
    fn try_accounts(
        __program_id: &Pubkey,
        __accounts: &mut &'info [AccountInfo<'info>],
        __ix_data: &[u8],
        __bumps: &mut InitializeBumps,
    ) -> anchor_lang::Result<Self> {
        let signer: Signer = anchor_lang::Accounts::try_accounts(
            ...
        ).map_err(...)?;

        let system_program: anchor_lang::accounts::program::Program<System> =
            anchor_lang::Accounts::try_accounts(
                ...
            ).map_err(...)?;

        Ok(Initialize {
            signer,
            system_program,
        })
    }
}
```

ackee
blockchain security

#[account]

- Handles the **Serialization** and **Deserialization** of the Account.
- Handles the **DISCRIMINATOR** creation and check.
- Defines the **Account owner** (i.e. your program).



ackee
blockchain security

Account Types

Account
- Serialization + Deserialization + Owner

Account Info
- -

Account Loader
- Zero copy deserialization

Boxed
- based on Inner Type

Interface
- Program Validation (from set)

Interface Account
- Serialization + Deserialization + Owner(from set)

Option
- based on Inner Type

Program
- Program Validation

Signer
- Signer Validation

System Account
- Owner (System Program)

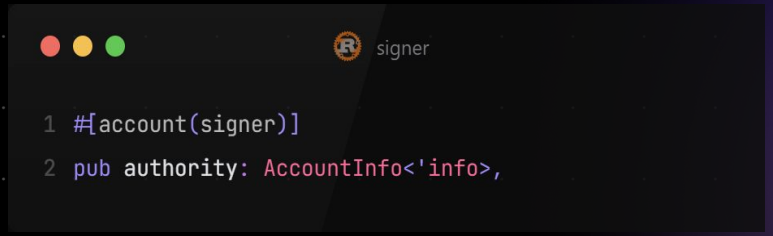
Sysvar
- Sysvar + Deserialization

Unchecked Account
- -

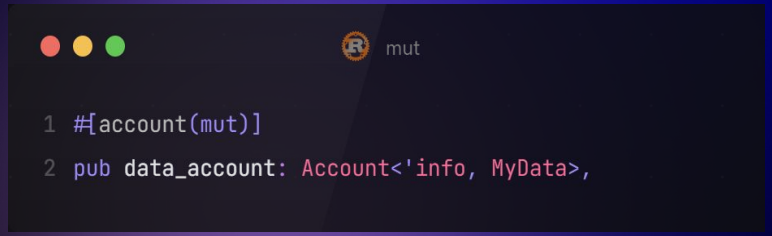
Account Types Docs

ackee
blockchain security

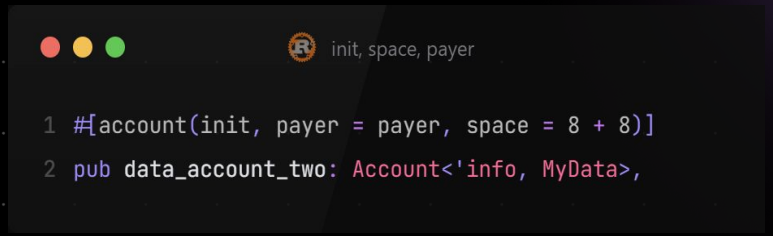
Account Constraints - Normal



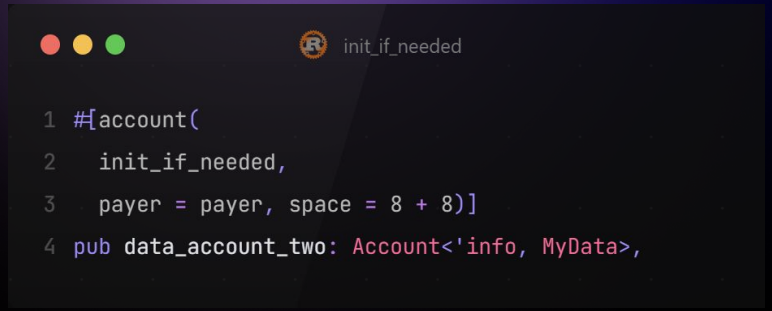
signer



mut



init, space, payer

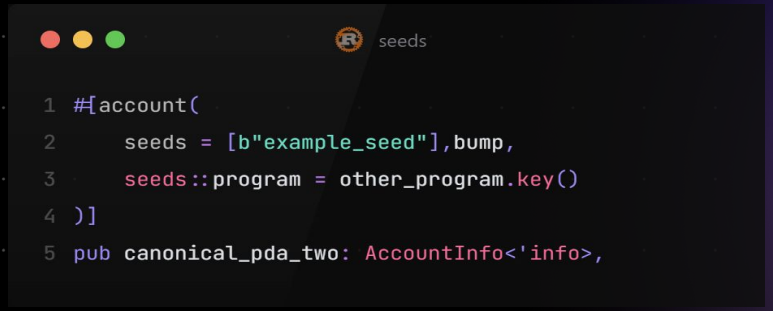


init_if_needed

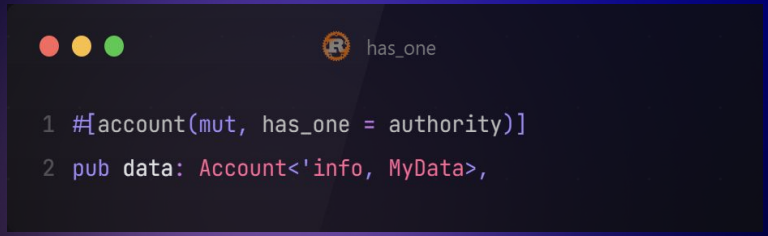
Account Constraints
Normal

ackee
blockchain security

Account Constraints - Normal

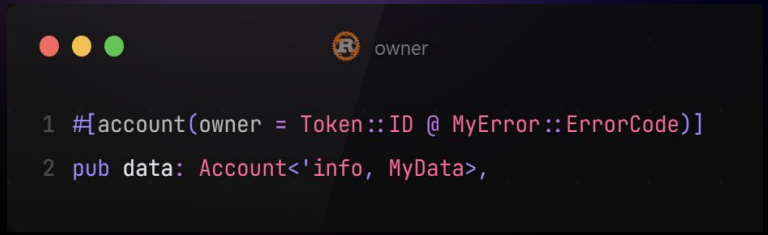


seeds



has_one

address



owner

Account Constraints
Normal

ackee
blockchain security

Account Constraints - Normal

executable

```
1 #[account(executable)]
2 pub my_program: AccountInfo<'info>
```

executable

rent_exempt

```
1 #[account(zero, rent_exempt = skip)]
2 pub skipped_account: Account<'info, MyData>,
```

rent_exempt

zero

```
1 #[account(zero)]
2 pub my_account: Account<'info, MyData>
```

zero

close

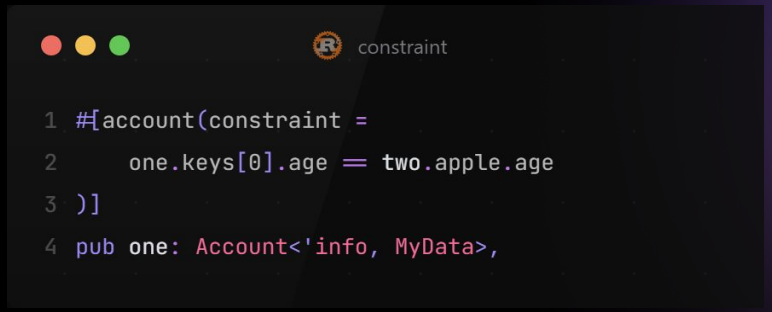
```
1 #[account(mut, close = receiver)]
2 pub data_account: Account<'info, MyData>,
```

close

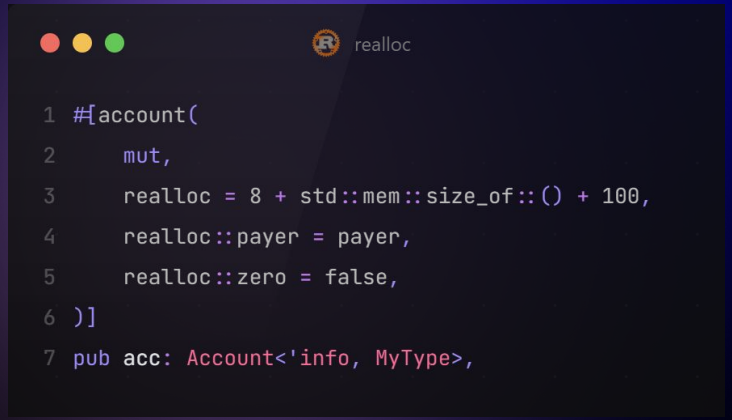
Account Constraints
Normal

ackee
blockchain security

Account Constraints - Normal



constraint



realloc

Account Constraints
Normal

ackee
blockchain security

Account Constraints - SPL

Account Constraints SPL

Token Account

Token Account

```
#[account(
    init,
    payer = payer,
    token::mint = mint,
    token::authority = payer,
)]
pub token: Account<'info, TokenAccount>,
```

Mint

Mint

```
#[account(
    init,
    payer = payer,
    mint::decimals = 9,
    mint::authority = payer,
    mint::freeze_authority = payer
)]
pub mint_two: Account<'info, Mint>,
```

Associated Token Account

Associated Token Account

```
#[account(
    init,
    payer = payer,
    associated_token::mint = mint,
    associated_token::authority = payer,
)]
pub token: Account<'info, TokenAccount>,
```

ackee
blockchain security

Anchor IDL

- The IDL follows a standardized JSON schema.
- The IDL file includes detailed information about each instruction in the program.
- The IDL provides a description of each account type used in the program.
- Front-end applications and other client tools can use the IDL to interact with the Solana program without needing to manually define the program's interface.

ackee
blockchain security

Recap

- #[program]
- #[derive(Accounts)]
- #[account]
- Anchor account Types
- Anchor account Constraints - Normal
- Anchor account Constraints - SPL
- IDL
- anchor expand

ackee.xyz

Thanks for your attention!

ackee
blockchain security

Email hello@ackee.xyz

Twitter @ackeeblockchain

Farcaster @ackee