

Solana Auditors Bootcamp

1. Intro & Advanced Anchor



ackee



Hi 🙌

Andrej, @andrey_19

Auditor @ Ackee Blockchain Security



@AckeeBlockchain
ackee.xyz



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



We work with



Grants



What is this course about ?

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

What this course is not about ?

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

Advanced Anchor Framework

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 ?

What is Vanilla Solana ?

Vanilla Solana



vanilla-solana.rs

```
1 // declare and export the program's entrypoint
2 entrypoint!(process_instruction);
3
4 // program entrypoint's implementation
5 pub fn process_instruction(
6     _program_id: &Pubkey,
7     _accounts: &[AccountInfo],
8     _instruction_data: &[u8],
9 ) -> ProgramResult {
10     // log a message to the blockchain
11     msg!("Hello, world!");
12
13     // gracefully exit the program
14     Ok(())
15 }
```

Anchor



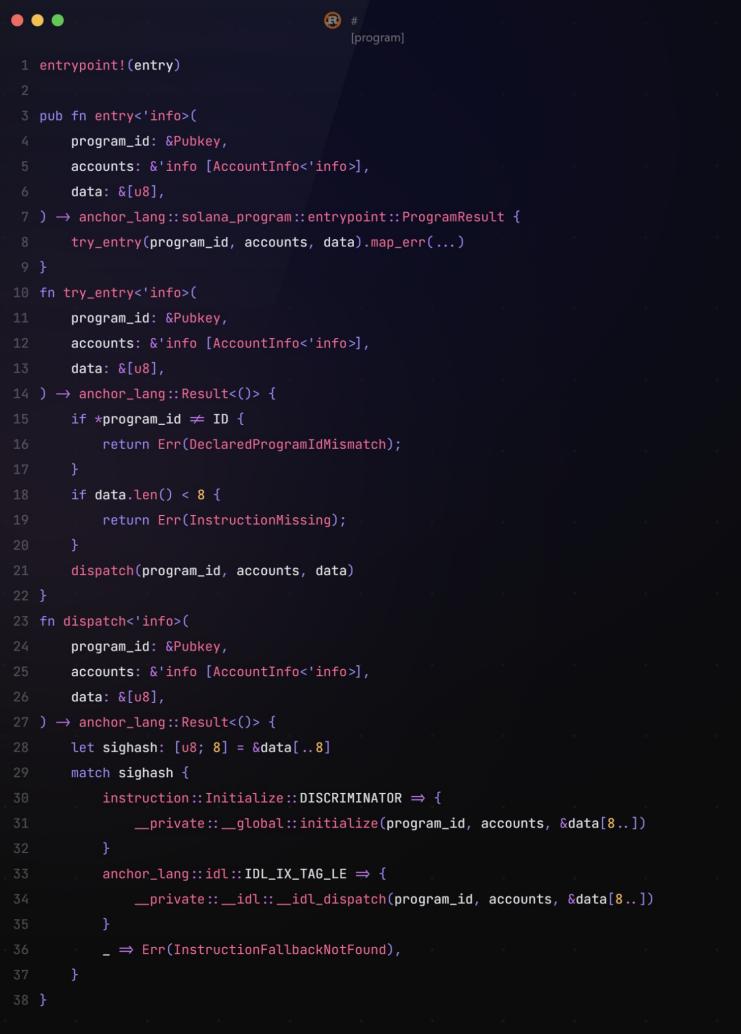
Anchor

```
1 use anchor_lang::prelude::*;
2
3 declare_id!("HrxxFK82k8J88kUvDZvTZHnqCVbeyjvebwfUQHwET4B7");
4
5 #[program]
6 pub mod anchor_solana {
7     use super::*;

8
9     pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
10         Ok(())
11     }
12 }
13
14 #[derive(Accounts)]
15 pub struct Initialize {}
```

#**[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)]***.



The screenshot shows a code editor window with a dark theme. The title bar says '# [program]'. The code is written in Rust and defines several functions related to Solana program logic:

```
1  entrypoint!(entry)
2
3  pub fn entry<'info>(
4      program_id: &Pubkey,
5      accounts: &'info [AccountInfo<'info>],
6      data: &[u8],
7  ) -> anchor_lang::solana_program::entrypoint::ProgramResult {
8      try_entry(program_id, accounts, data).map_err(...)
9  }
10 fn try_entry<'info>(
11     program_id: &Pubkey,
12     accounts: &'info [AccountInfo<'info>],
13     data: &[u8],
14 ) -> anchor_lang::Result<()> {
15     if *program_id != ID {
16         return Err(DeclaredProgramIdMismatch);
17     }
18     if data.len() < 8 {
19         return Err(InstructionMissing);
20     }
21     dispatch(program_id, accounts, data)
22 }
23 fn dispatch<'info>(
24     program_id: &Pubkey,
25     accounts: &'info [AccountInfo<'info>],
26     data: &[u8],
27 ) -> anchor_lang::Result<()> {
28     let sighash: [u8; 8] = &data[..8]
29     match sighash {
30         instruction::Initialize::DISCRIMINATOR => {
31             __private::__global::__initialize(program_id, accounts, &data[8...])
32         }
33         anchor_lang::idl::IDL__IX_TAG_LE => {
34             __private::__idl::__idl_dispatch(program_id, accounts, &data[8...])
35         }
36         _ => Err(InstructionFallbackNotFound),
37     }
38 }
```

#[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)]  
  
1 impl<'info> anchor_lang::Accounts<'info, InitializeBumps> for Initialize<'info>  
2 where  
3     'info: 'info,  
4 {  
5     fn try_accounts(  
6         __program_id: &Pubkey,  
7         __accounts: &mut '&info [AccountInfo<'info>],  
8         __ix_data: &[u8],  
9         __bumps: &mut InitializeBumps,  
10    ) -> anchor_lang::Result<Self> {  
11        let signer: Signer = anchor_lang::Accounts::try_accounts(  
12            ...  
13        ).map_err(...)?;  
14  
15        let system_program: anchor_lang::accounts::program::Program<System> =  
16            anchor_lang::Accounts::try_accounts(  
17                ...  
18            ).map_err(...)?;  
19  
20        Ok(Initialize {  
21            signer,  
22            system_program,  
23        })  
24    }  
25 }
```

#[account]

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

```
#[account]

1 #[derive(AnchorSerialize, AnchorDeserialize, Clone)]
2 pub struct DataAccount {
3     pub authority: Pubkey,
4 }
5
6 impl anchor_lang::AccountSerialize for DataAccount {
7     fn try_serialize<W: std::io::Write>(&self, writer: &mut W) → anchor_lang::Result<()> {
8         if writer
9             .write_all(&[85, 240, 182, 158, 76, 7, 18, 233])
10            .is_err()
11        {
12             return Err(AccountDidNotSerialize.into());
13         }
14         if AnchorSerialize::serialize(self, writer).is_err() {
15             return Err(AccountDidNotSerialize.into());
16         }
17         Ok(())
18     }
19 }
20 impl anchor_lang::AccountDeserialize for DataAccount {
21     fn try_deserialize(buf: &mut &[u8]) → anchor_lang::Result<Self> {
22         if buf.len() < [85, 240, 182, 158, 76, 7, 18, 233].len() {
23             return Err(AccountDiscriminatorNotFound.into());
24         }
25         if &[85, 240, 182, 158, 76, 7, 18, 233] ≠ &buf[..8]{
26             return Err(AccountDiscriminatorMismatch);
27         }
28         Self::try_deserialize_unchecked(buf)
29     }
30     fn try_deserialize_unchecked(buf: &mut &[u8]) → anchor_lang::Result<Self> {
31         AnchorDeserialize::deserialize(&buf[8..]).map_err(AccountDidNotDeserialize)
32     }
33 }
```



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



Account Constraints - Normal



```
1 #[account(signer)]  
2 pub authority: AccountInfo<'info>,
```



signer



mut

signer

mut



init, space, payer

```
1 #[account(init, payer = payer, space = 8 + 8)]  
2 pub data_account_two: Account<'info, MyData>,
```



init_if_needed

```
1 #[account(  
2     init_if_needed,  
3     payer = payer, space = 8 + 8)]  
4 pub data_account_two: Account<'info, MyData>,
```

init, space, payer

init_if_needed



Account Constraints - Normal

● ● ●

seeds

```
1 #[account(  
2     seeds = [b"example_seed"], bump,  
3     seeds :: program = other_program.key()  
4 )]  
5 pub canonical_pda_two: AccountInfo<'info>,
```

seeds

● ● ●

has_one

```
1 #[account(mut, has_one = authority)]  
2 pub data: Account<'info, MyData>,
```

has_one

● ● ●

address

```
1 #[account(address = crate::ID)]  
2 pub data: Account<'info, MyData>,
```

address

● ● ●

owner

```
1 #[account(owner = Token::ID @ MyError::ErrorCode)]  
2 pub data: Account<'info, MyData>,
```

owner

Account Constraints
Normal



Account Constraints - Normal



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



executable



rent_exempt

executable

rent_exempt



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



zero



close

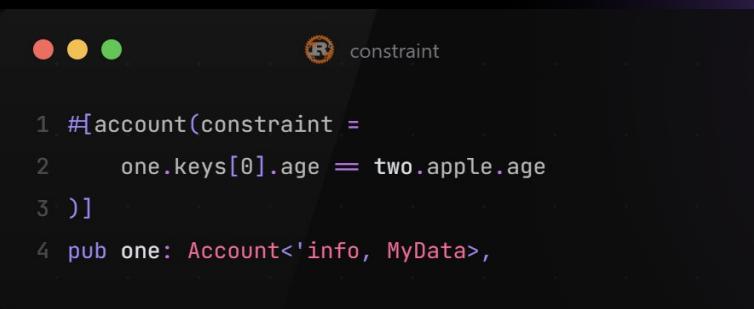
zero

close

Account Constraints
Normal



Account Constraints - Normal



```
● ● ● constraint

1 #[account(constraint =
2     one.keys[0].age == two.apple.age
3 )]
4 pub one: Account<'info, MyData>,
```

constraint



```
● ● ● realloc

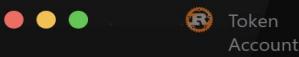
1 #[account(
2     mut,
3     realloc = 8 + std::mem::size_of() + 100,
4     realloc::payer = payer,
5     realloc::zero = false,
6 )]
7 pub acc: Account<'info, MyType>,
```

realloc



Account Constraints - SPL

Token Account



```
1 #[account(
2     init,
3     payer = payer,
4     token::mint = mint,
5     token::authority = payer,
6 )]
7 pub token: Account<'info, TokenAccount>,
```

Mint


```
1 #[account(
2     init,
3     payer = payer,
4     mint::decimals = 9,
5     mint::authority = payer,
6     mint::freeze_authority = payer
7 )]
8 pub mint_two: Account<'info, Mint>,
```

Associated Token Account


```
1 #[account(
2     init,
3     payer = payer,
4     associated_token::mint = mint,
5     associated_token::authority = payer,
6 )]
7 pub token: Account<'info, TokenAccount>,
```



Anchor IDL

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.

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!



Email hello@ackee.xyz

Twitter [@ackeeblockchain](#)

Farcaster [@ackee](#)