

Anatomy of a Solana program

Reading a smart contract the way an attacker does

THE PROGRAM

A tip vault, about 90 lines of Rust written with Anchor.

THE PROMISE

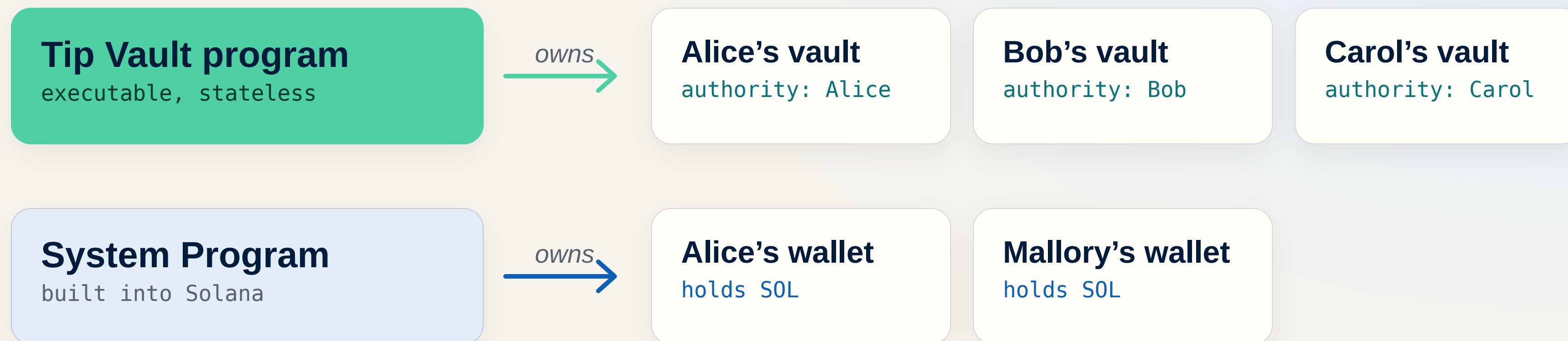
Anyone can tip into a vault. Only its owner can withdraw.

OUR JOB

Find the exact lines that make that promise true, then break one.



Programs hold code. Accounts hold everything else.



The runtime's one hard rule: **only an account's owner program can take lamports out of it.** Everything else is up to the program's own checks.

The caller chooses which **accounts** to pass in

Every Solana instruction lists every account it will touch before it runs.

That list is why Solana is fast: transactions that touch different accounts can run in parallel.

It also means the attacker writes the list. **A program that trusts an account just because it showed up has already lost.**

So every program has to ask three questions:

1

Did this account sign?

`Signer<'info>`

2

Is it really ours, and the right type?

`Account<'info, Vault>`

3

Do these accounts belong together?

`has_one and seeds`

What to watch for **in the demo**

How Solana works differently

- The program ID and its upgrade authority
- A module with no storage variables
- Account lists declared up front
- PDAs: addresses with no private key
- Paying rent for every byte stored
- Calling the System Program to move SOL

Security checks

- Owner and type checks on the vault
- The Signer on the withdrawing account
- has_one and seeds on Withdraw
- Checked math and the rent floor

Predict: if we delete the constraints from Withdraw, who can take the money?

Write down your answer. We'll check it in about ten minutes.