Solidity Programming

A Five-Minute Overview

BSc Blockchain Course

Why Bugs in Smart Contracts Cost Billions

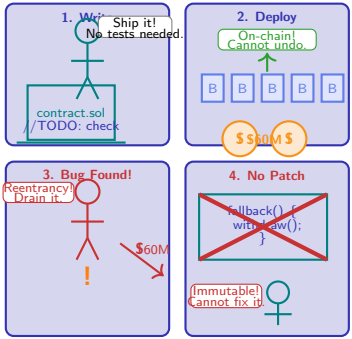
Smart contracts run on a public blockchain: once deployed, **no one can patch them**. A bug discovered on day 2 is as permanent as the contract itself.

What makes this different from normal software:

- **Immutability** – deployed bytecode cannot be changed
- **Value at stake** – contracts often hold millions of dollars
- **No hotfix** – you cannot push a patch at 2am
- **Solution** – audit first, deploy once, deploy right

The DAO Hack (2016):
A reentrancy bug let an attacker drain **$**60 M in ETH before anyone could respond. The only fix was a hard fork of the entire Ethereum blockchain.

"Move fast and break things" is the wrong philosophy when things hold other people's money.



Immutability is a feature for trustless execution but a liability for buggy code. The discipline of Solidity development means front-loading every quality check before the contract ever touches the blockchain.

Sourc

Mehar et al. (2019). "Understanding a Revolutionary and Flawed Grand Experiment." Journal of Cases on IT, 21(1). DAO hack: $60M ETH, June 2016.

What Are the Six Layers Inside Every Solidity Contract?

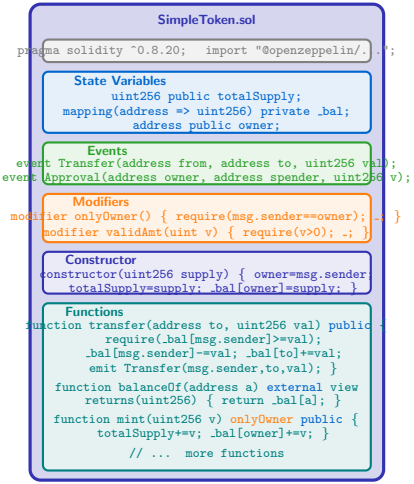
Every Solidity file shares the same layered structure. Understanding this anatomy lets you navigate any contract on Etherscan instantly.

The six sections, top to bottom:

- **Pragma & Imports** – declare compiler version and pull in libraries
- **State Variables** – persistent on-chain storage; every write costs gas
- **Events** – off-chain logging; front-ends listen for these
- **Modifiers** – reusable access guards applied to functions
- **Constructor** – runs once on deployment; initialises state
- **Functions** – external/public/internal/private; the behaviour layer

Visibility ladder (most open to most restricted):

`public`	callable by anyone, from anywhere
`external`	callable only from outside the contract
`internal`	callable by this contract and derived ones
`private`	callable only by this contract



Reading any unfamiliar contract top-to-bottom in this order – pragma, state, events, modifiers, constructor, functions – reveals its full behaviour without needing to run it.

Sourc

Why Must You Update State Before Sending Money?

The most dangerous smart contract pattern is reentrancy: an attacker's contract calls back into your function before you have finished updating state. The fix is a three-step ordering rule applied to *every* function that moves value.

Step 1 – CHECKS: validate all inputs and preconditions

- `require(balances[msg.sender] >= amount);`
- `require(amount > 0);`

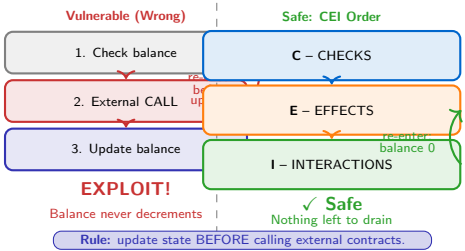
Step 2 – EFFECTS: update all contract state *before* any call

- `balances[msg.sender] -= amount;` ✓ update first

Step 3 – INTERACTIONS: only now call external contracts

- `(bool ok,) = msg.sender.call{value: amount}("");`
- Even if the callee re-enters, the balance is already zero

The DAO (2016): skipped Step 2. Balance was decremented *after* the external send. Attacker looped the send before the decrement ever ran. Result: $60 M drained.



1. Check balance

CEI is the primary defence. When CEI ordering is structurally difficult, OpenZeppelin's `ReentrancyGuard` (`nonReentrant` modifier) provides a boolean-mutex fallback – but CEI should always be applied first.

Sourc

How Does Solidity Source Code Become an On-Chain Contract?

Deploying a Solidity contract is a four-stage pipeline. Each stage transforms the artifact into something closer to what the EVM can execute and verify.

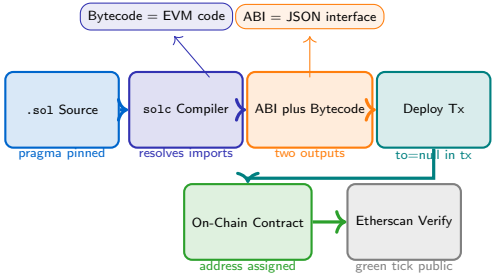
Stage 1 – Write (`.sol`):
Human-readable Solidity. Compiler version pinned via `pragma`. Imports from OpenZeppelin resolve at compile time.

Stage 2 – Compile (`solc`):
`solc` or Hardhat's built-in compiler produces two outputs: the **ABI** (the public interface JSON) and the **bytecode** (the EVM machine code).

Stage 3 – Deploy transaction:
Send a transaction with `data = bytecode` and `to = null`. The EVM executes the bytecode, runs the constructor, and assigns the contract address.

Stage 4 – Verify on Etherscan:
Upload the source code to Etherscan. It re-compiles and checks the bytecode matches. Verified contracts show a green tick and the source to anyone.

After Stage 4, wallets and front-ends use the ABI to call functions by name.



The ABI is the contract's public API: it tells wallets and front-ends the function names, parameter types, and return values without them having to decode raw bytecode.

Sourc

Ethereum Yellow Paper (Wood, 2014). Hardhat Documentation (hardhat.org). Etherscan contract verification docs.

Six Rules That Separate Secure Contracts from Expensive Mistakes

Before any smart contract goes to mainnet, walk through six questions. Each maps to a class of real-world exploit.

- ✓ Checks-Effects-Interactions ordering on all value functions
- ✓ Visibility defaults `private`/`internal`; `public`/`external` only where required
- ✓ No raw `tx.origin` for access control; use `msg.sender`
- ✓ Unit tests cover all revert conditions, not just happy paths
- ✓ Fuzz tests applied to value-handling functions; independent audit before deploy
- ✓ On-chain monitoring active; pause/upgrade mechanism is time-locked

Why each item is on the list:

- **CEI** – prevents reentrancy (**$**60M DAO, **$**25M Cream Finance)
- **Minimal visibility** – removes attack surface; private state cannot be called directly
- **msg.sender not tx.origin** – `tx.origin` can be manipulated via phishing contracts
- **Revert tests** – happy-path-only tests miss the conditions attackers target
- **Fuzzing** – discovers integer edge cases that unit tests rarely probe
- **Audit** – Trail of Bits, ConsenSys Diligence, Code4rena: worth every dollar before a **$**10M launch
- **Monitoring** – exploits often unfold over multiple transactions; alerts buy response time
- **Time-locked upgrades** – prevents a compromised admin key from silently draining funds overnight

Each item on this checklist maps directly to a class of attack that has cost users hundreds of millions of dollars.

Sourc

ConsenSys Diligence. “Smart Contract Best Practices.” consensys.github.io/smart-contract-best-practices. OpenZeppelin Security Audits 2023.