Security Best Practices

Patterns, testing and chain-specific cautions for contracts that hold real value.

VERSION	LAST UPDATED	STATUS
0.1	November 17, 2026	Draft

1. Principles

Smart contracts are public, immutable by default and hold value directly, so mistakes are expensive and usually cannot be undone. Keep contracts small, reuse audited libraries, minimise privileged roles and test far more than you think you need to.

> The examples in these guides are for learning. They are not audited and are not financial, legal or security advice. Have any contract that will hold real value reviewed by an independent auditor before launch.

2. Common vulnerabilities

Risk	What goes wrong	Mitigation
Reentrancy	An external call lets an attacker call back into your contract before state is updated	Checks-effects-interactions and ReentrancyGuard
Broken access control	A privileged function is open to everyone or to the wrong role	Explicit roles, onlyOwner or AccessControl, and tests for every restricted function
Unchecked external calls	A failed call is ignored and state becomes inconsistent	Check return values and use SafeERC20
Price manipulation	A spot price is read from a pool that an attacker can move in one transaction	Use time-weighted prices or a trusted oracle with sanity checks

Signature replay	A valid signature is reused on another chain, contract or time	EIP-712 domain with chain ID and contract, nonces and deadlines
Unsafe upgrades	Storage layout changes corrupt state, or an initializer can be called twice	Upgrade safety checks and protected initializers
Denial of service	An unbounded loop or one failing recipient blocks everyone	Pull payments and bounded iteration

3. Checks, effects, interactions

Validate inputs first, update your own state second and call external contracts last. Add a reentrancy guard on top for functions that move value.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

contract Vault is ReentrancyGuard {
using SafeERC20 for IERC20;

IERC20 public immutable asset;
mapping(address => uint256) public balances;

constructor(IERC20 asset_) {
asset = asset_;
}

function deposit(uint256 amount) external nonReentrant {
require(amount > 0, "zero amount");
balances[msg.sender] += amount;
asset.safeTransferFrom(msg.sender, address(this), amount);
}

function withdraw(uint256 amount) external nonReentrant {
require(balances[msg.sender] >= amount, "insufficient balance");
balances[msg.sender] -= amount;
asset.safeTransfer(msg.sender, amount);
}
}
```

4. Chain-specific cautions

The EVM is the same, but the chain around it is not. These are the details that most often break contracts when moving between networks.

- Do not hardcode the chain ID, block time or block number arithmetic. Read block.chainid and use block.timestamp for time.
- Do not assume a contract exists at a well-known address. Wrapped native token, Multicall3, Permit2 and CREATE2 deployers may be missing or live elsewhere. Check with cast code on this network.
- Check which precompiles and opcodes the network supports, and compile with a matching evm_version. A too-new target can produce bytecode the network rejects.
- Gas costs and the block gas limit can differ. Do not rely on a fixed gas amount for external calls, and test gas-heavy functions on the testnet.
- Do not assume any L1 data is available. Features that rely on cross-chain messaging need explicit network support: [cross-chain messaging availability].
- If an oracle or price feed is required, confirm that it is live on this network before launch: [oracle availability].

5. Testing

- Unit tests for every function, including every revert path.
- Fuzz tests for any function that takes numbers or addresses.
- Invariant tests for properties that must always hold, such as total supply equals the sum of balances.
- Fork or testnet tests against the real network, not only a local chain.
- Static analysis with Slither on every commit, and review every finding.
- An independent audit before mainnet, and a re-audit after any significant change.

```
forge test --fuzz-runs 5000
forge coverage
slither .
```

6. Key management and operations

- Use a hardware wallet or multisig for every privileged role. Never keep an owner key in a repository, a chat or a shared document.
- Separate the deployer, the admin and the day-to-day operator roles.
- Add a time delay for sensitive admin actions so users can react.
- Monitor large transfers and admin events and alert a human.

- Prepare an incident plan: who can pause, who decides, and how users are told.

Network security contact: [security contact]. Bug bounty program: [bug bounty URL].

This document is for information only and is not financial, legal or tax advice. Highlighted items in square brackets are still to be confirmed by the issuer. The latest version is always available at neosmartchain.com/developers/security