Pre-launch smart contract checklist

40 checks before mainnet, from a team with 700+ deployed contracts.

By Usman Ul Haq, founder · x.com/EtherDev · Updated November 2026

IF YOU ONLY CHECK FIVE THINGS

1. Who can call every admin function?
2. Does the math round the way you expect?
3. Does state change before every external call?
4. Can signatures be faked or reused?
5. Does it pass fuzz and invariant tests on a mainnet fork?

Written for Solidity contracts on EVM chains such as Ethereum, BNB Chain, Base, Arbitrum and Polygon. Tick each box once the code is frozen, then again after your testnet dry run.

01 Access control

Many of the worst exploits come down to someone calling a function they shouldn't be able to.

- [ ] **Every state-changing function has the right access check**
  List every `public` and `external` function that changes state. Each one should be open to everyone on purpose, or protected by `onlyOwner` or a role.
- [ ] **Ownership sits in a multisig**
  Move ownership to a multisig before launch, not a single deployer wallet whose key sits on one laptop.
- [ ] **Ownership transfers take two steps**
  Use `Ownable2Step` or similar, so ownership can't be sent to a mistyped address.
- [ ] **Initializers can only run once**
  In upgradeable contracts, protect `initialize`, and call `_disableInitializers()` in the implementation's constructor.
- [ ] **Every admin power is written down**
  List what the owner can do (mint, pause, change fees, upgrade, withdraw) and who holds each key. Holders will ask, so publish it.

02 Math and logic

Solidity has no decimals, and integer division rounds down. Small rounding mistakes become real losses at scale.

- [ ] **Multiply before you divide**
  `score / maxScore * 100` is 0 unless the score reaches the max. Write `score * 100 / maxScore`.

- [ ] **Rounding works in the protocol's favor**
  When splitting rewards or shares, round down what users receive, and make sure leftover dust can't be drained or block withdrawals.
- [ ] **Token decimals are read, not assumed**
  Stablecoins often use 6 decimals on Ethereum and 18 on BNB Chain. Read `decimals()` instead of hardcoding `1e18`.
- [ ] **Edge cases are tested**
  Zero amounts, the first and last user, maximum values, and an empty pool or reward list.
- [ ] **Every `unchecked` block is justified**
  Solidity 0.8 reverts on overflow, except inside `unchecked`. Each one needs a clear reason it can't overflow.

03 External calls

Any call to another contract hands control to code you didn't write.

- [ ] **State changes before external calls**
  Update balances and ownership first, then call out or send ETH (checks-effects-interactions). Calling out first is how reentrancy happens.
- [ ] **A reentrancy guard where funds move**
  Add `nonReentrant` to functions that send ETH or tokens, or that call an address the user supplies.
- [ ] **Token transfers use SafeERC20**
  Some tokens, like USDT on Ethereum, don't return a bool. `safeTransfer` handles them and reverts on failure.
- [ ] **Received amounts are measured, not assumed**
  Fee-on-transfer tokens deliver less than the amount sent. If you accept any token, compare the balance before and after.
- [ ] **Prices can't be moved in one transaction**
  Never price off a single DEX pool's current reserves, because a flash loan can move them. Use a TWAP or an oracle like Chainlink, and check how fresh the price is.

04 Signatures

If your app signs messages for the contract to accept, the contract has to check them properly.

- [ ] **The signer is checked**
  Recover the signer and compare it to your app's signing address. A function like `addPoints(user, pts)` with no signature lets anyone award themselves points.
- [ ] **Each signature can only be used once**
  Include a nonce, or mark each signature as used, so the same one can't be submitted twice.
- [ ] **Signatures expire**
  Include a deadline in the signed data and reject anything past it.
- [ ] **Signatures only work on this contract and chain**
  Use EIP-712 with the chain ID and contract address in the domain, so a testnet signature, or one meant for another contract, fails here.

- [ ] **Invalid signatures revert**
  Raw `ecrecover` returns `address(0)` for a bad signature, which passes if the expected signer is ever unset. OpenZeppelin's ECDSA library reverts instead.

05 Token settings

For token launches. Holders and token scanners read these settings before they buy.

- [ ] **Supply and allocations match what you announced**
  Check minted amounts, decimals and every wallet allocation against your published tokenomics.
- [ ] **Fees and limits have hard caps**
  If the owner can change buy and sell fees or the max wallet size, cap them in the contract so they can never be set to 100%.
- [ ] **Trading can't be switched off again**
  Once trading is enabled, there should be no way to disable it. A switch that can stop sells looks like a honeypot to holders and scanners.
- [ ] **No hidden mint, blacklist or fee exemptions**
  Scanners will find any mint, blacklist or fee-exempt function. Remove it, or explain publicly why it's there.
- [ ] **The liquidity plan is in the launch script**
  Decide whether LP tokens are locked or burned, and do it in the script so it isn't forgotten on launch day.

06 Testing

Coverage shows which lines ran, not whether they're right. Test what must never happen, not just what should.

- [ ] **Unit tests for every function and every revert**
  Test the failure paths, not just the happy path.
- [ ] **Fuzz tests on inputs**
  Let Foundry try thousands of random amounts and addresses, for example `testFuzz_strangerCannotBecomeOwner(address)`.
- [ ] **Invariant tests for what must always be true**
  For example: total supply never exceeds the cap, and the contract always holds enough to pay every claim.
- [ ] **A fork test against mainnet**
  Run the launch on a fork of the real chain, with the real router, tokens and oracles.
- [ ] **Coverage reviewed, not just measured**
  `forge coverage` shows lines that never ran. Read what's missing instead of chasing 100%.

07 Deployment

Most launch-day problems are settings, not code.

- [ ] **A testnet dry run with the real script**
  Deploy with the exact script and settings you'll use on mainnet, then click through the whole app.

- [ ] **Constructor arguments checked twice**
  Owner, token addresses, router, fees and start times. A wrong address here usually means redeploying.
- [ ] **Source code verified on the block explorer**
  Use `forge create --verify` or `forge verify-contract`. Holders treat unverified contracts as a red flag.
- [ ] **Compiler version and settings pinned**
  Use a fixed `pragma` and the same optimizer settings you tested with.
- [ ] **The deployer key is safe**
  Deploy from a fresh wallet or a hardware wallet, and never put a private key in a file that gets committed.

08 After launch

Plan for problems before you have one.

- [ ] **Monitoring and alerts are on**
  Get alerts for large transfers, admin calls and failed transactions, so you hear about problems before your holders do.
- [ ] **There's a pause switch, and you know who holds it**
  For contracts that hold funds, a `pause` function on a multisig buys time during an incident.
- [ ] **Admin changes go through a timelock**
  For contracts holding user funds, a delay on upgrades and setting changes gives holders time to react.
- [ ] **An incident plan is written down**
  Who gets called, how you pause and what you post publicly. Decide it before you need it.
- [ ] **Official addresses are published in one place**
  List every contract address on your site and socials so holders can spot fakes.

Three bugs that pass your tests

Tests check what you expected to happen. These bugs live in what you didn't expect, and each one fits in a single line.

BUG 1

No signature check

A function your app is meant to call, but anyone can call it and award themselves points.

```
- function addPoints(address user, uint pts) external {
+ function addPoints(address user, uint pts, bytes calldata sig) external {
+   require(verify(user, pts, sig), "not signed by app");
```

BUG 2

Dividing before multiplying

Integer division rounds down first, so the percentage comes out as 0 for almost every player.

```
- uint pct = score / maxScore * 100;  // 0 unless score ⩾ maxScore
+ uint pct = score * 100 / maxScore;
```

BUG 3

Calling out before updating state

The receiver can call back in before ownership is recorded, and act on the old state.

```
- IReceiver(to).onReceive(id);
- owner[id] = to;
+ owner[id] = to;
+ IReceiver(to).onReceive(id);
```

Want a second pair of eyes before launch?

Send us your contracts and we'll go through them against this checklist before you deploy.

Get a free review or message @EtherDev on X

This checklist doesn't replace an audit. It covers the common mistakes. If a contract will hold significant user funds, get an independent audit as well. Online version: infinityblockchainsolutions.com/smart-contract-launch-checklist