What is a contract DoS?

A smart contract denial of service (DoS) makes a function or an entire contract unusable, sometimes permanently, which can lock funds forever. Unlike traditional DoS, it is usually a logic or design flaw: a loop over an unbounded array that runs out of gas, a payment to an address that always reverts (blocking a queue), or a privileged role that gets stuck. Because contracts are immutable, a DoS bug can be unrecoverable. The fix is pull-over-push payments, bounded loops, and no single point that can block everyone.

HOW IT WORKS

01 How it works and example

Common on-chain DoS patterns:

- Unbounded loop / gas exhaustion: a function iterates over an array anyone can grow (for example a list of participants). Once it is large enough, the loop exceeds the block gas limit and the function can never complete.
- Revert-based DoS (push payments): a contract pays out by pushing funds in a loop; one recipient is a contract that always reverts, so the whole distribution fails for everyone.
- Stuck privileged role: an owner-only step where the owner key is lost or set to a contract that cannot act, freezing the protocol.

Documented for defensive context.

> IMMUTABLE PLUS STUCK EQUALS LOST
>
> *A DoS that blocks a fund-controlling function on an immutable contract can lock value permanently. The classic cause is push payments in a loop, one reverting recipient blocks everyone; use pull payments instead.*

HOW TO DEFEND

- Use pull-over-push payments: let each user withdraw their own funds, so one bad recipient cannot block others.
- Avoid unbounded loops over user-growable arrays; cap iteration or use pagination/withdrawal patterns.
- Never let one external call's failure halt a process for everyone; isolate failures.
- Avoid single points of control that can get stuck; use multisig and recovery paths for privileged steps.
- Audit and gas-test functions against worst-case sizes and reverting counterparties.

SOURCES

[1] SWC Registry: Smart Contract Weakness Classification

[2] Solidity docs: Security considerations

[3] MITRE CWE-400: Uncontrolled Resource Consumption