Block Stuffing

Explainer

Block stuffing is a denial-of-service or manipulation attack where a malicious actor deliberately **fills the available gas limit of a block** with spam or low-value transactions, **preventing or delaying other critical transactions** from being included.

> Think of it as “spamming the blockchain to block others’ moves.”

This can be used to:

- Prevent others from executing arbitrage trades
- Block liquidation bots from acting
- Force your own Tx to execute at a better price (MEV)
- Delay oracle updates or governance proposals
- Attack protocols with **tight time windows**

Cause

- Ethereum and other EVM chains have **block gas limits**, meaning there’s a maximum amount of computation (gas) that fits in one block.
- If a block is full, **other transactions must wait**.
- Attackers **send cheap, high-gas transactions** to artificially fill the block.
- Combined with fast miners or priority fee bumping, this attack can be repeated across multiple blocks.

Where to Look

Check for:

- **Time-sensitive operations** that must happen **within X blocks**
- **Auctions** or **flash loan windows** that last 1 block or a few seconds
- **Liquidation windows** where delay increases loss
- **Oracles** that need to update each block or read block timestamps
- **Governance votes** or staking locks that depend on specific block numbers

> Any place where **“must happen quickly or fairly”** is a core assumption.

Why This Happens

Because:

- Ethereum allows **anyone to submit transactions**, and miners/validators can include them.
- Block inclusion is **not guaranteed** even for valid transactions — it depends on gas price and block fullness.
- Many protocols rely on assumptions like “someone will call this function if X happens,” but an attacker can **prevent that** from happening at the right time.

Recommended Solutions

Design-Level Mitigations:

1. **Don’t Depend on 1-Block Windows**
   - Avoid actions that must be performed *immediately* or *within one block*
2. **Add Execution Buffers**
   - Allow for **multi-block grace periods** for time-critical operations
3. **Automate Critical Functions via Bots or Keepers**
   - Integrate with reliable automation networks (e.g., Chainlink Keepers, Gelato)
4. **Use Priority Queues**
   - If multiple actors can act, reward the **first successful call**, not the fastest
5. **Incentivize Redundancy**
   - Let multiple participants try to execute, only rewarding the first success
6. **Design for MEV Resistance**
   - Use commit-reveal or batch processing to reduce timing-based attacks