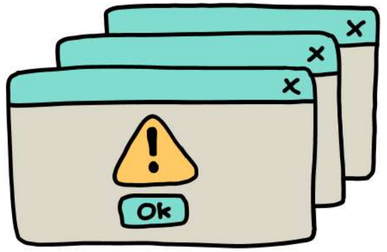


Inside the Biggest Outages: Hidden Failure Patterns Every SRE Should Know

Unnati Mishra

R & D Engg💻 @Broadcom

CKA, CKAD 📜

Public Speaker 🎤

Today’s Agenda

01 The Pattern Problem: Why major outages keep happening

02 Case Studies: CrowdStrike, Cloudflare, AWS

03 Quick overview of what happened and
Deep dive into hidden technical patterns

04 Demo

05 Key Takeaways

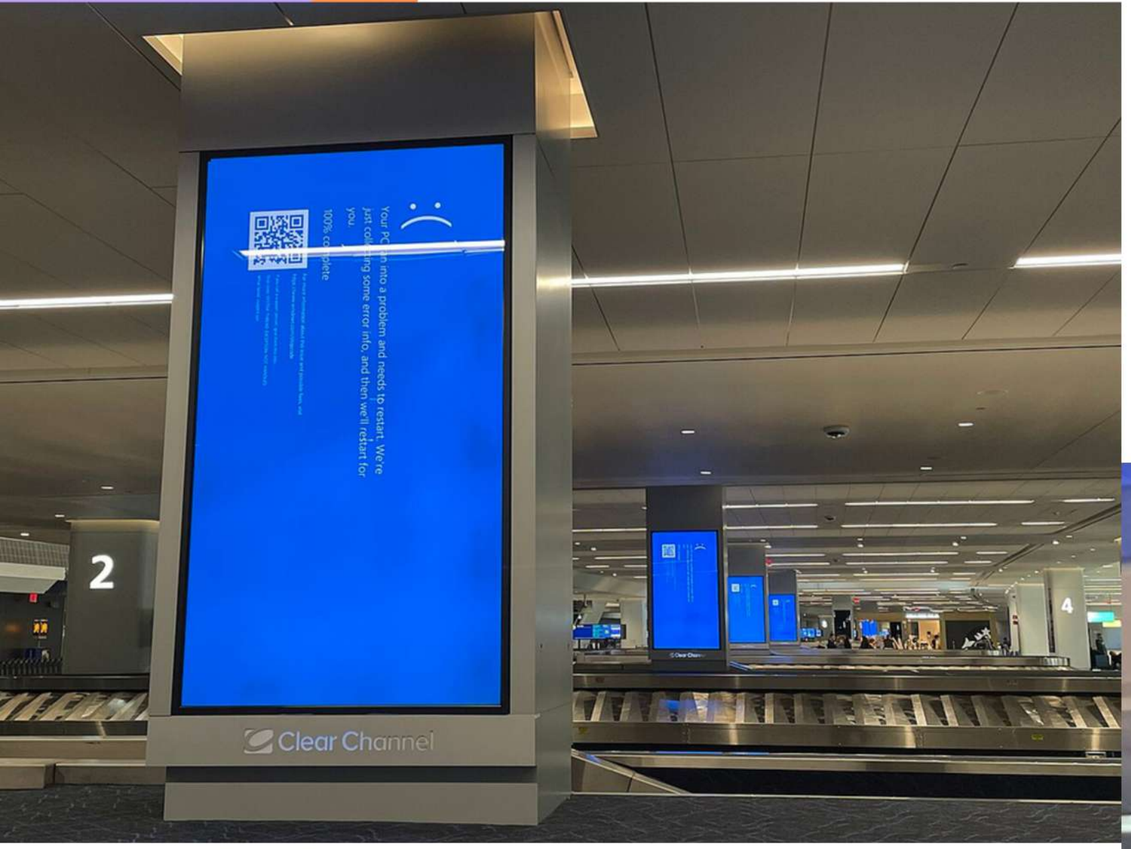
The Pattern Problem - Why Major Outages Keep Happening

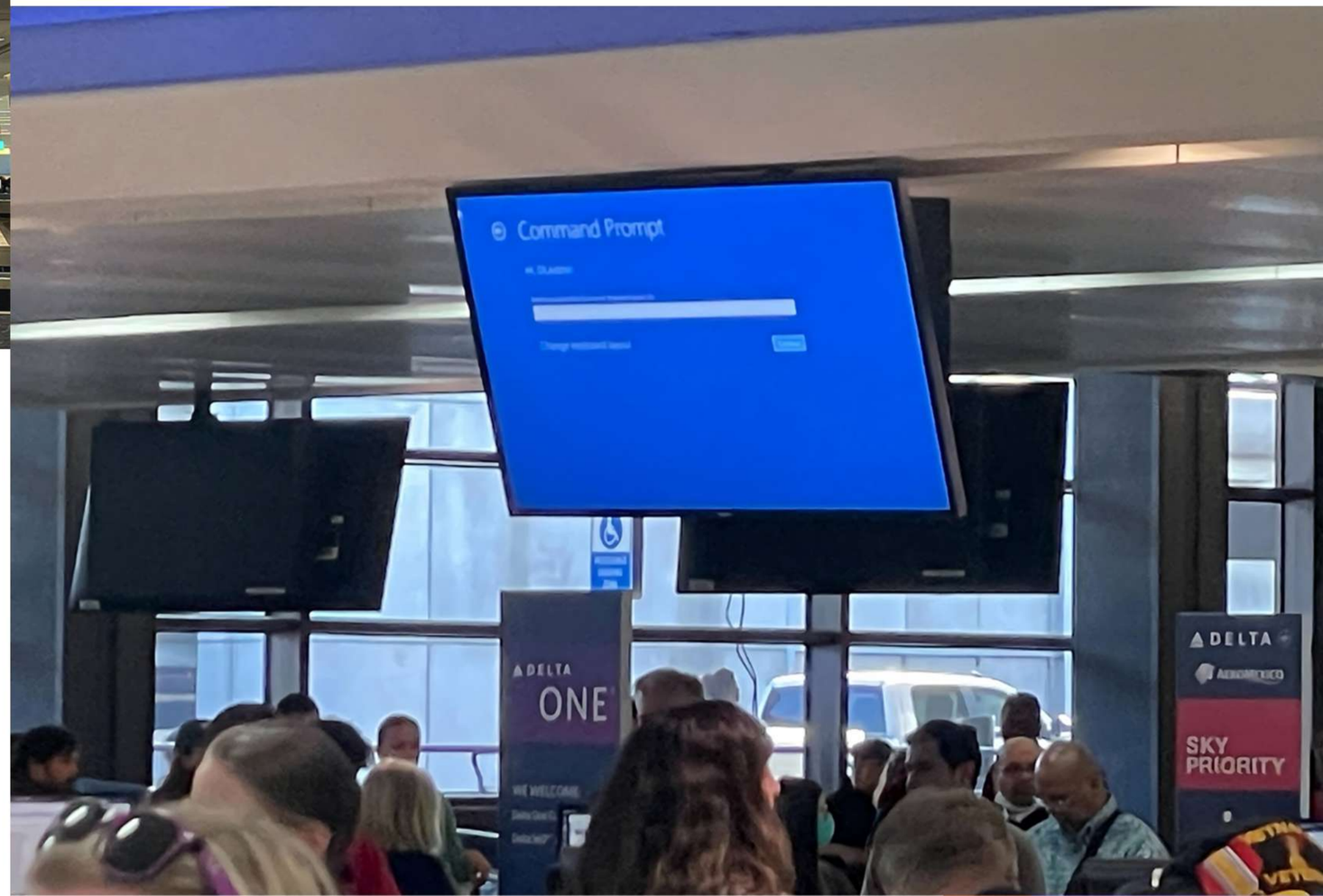
- **2024-2025**: Record number of planetary-scale outages
- **The Question**: Are systems getting worse? Or are we just seeing failures more clearly?
- **The Uncomfortable Truth**: Neither. We're repeating the same patterns.

Why These Outages Matter

- Wide reach due to centralization
- Hidden fragility
- Cascading real-world impact
- Rise in frequency

Incident Overview - The Basics



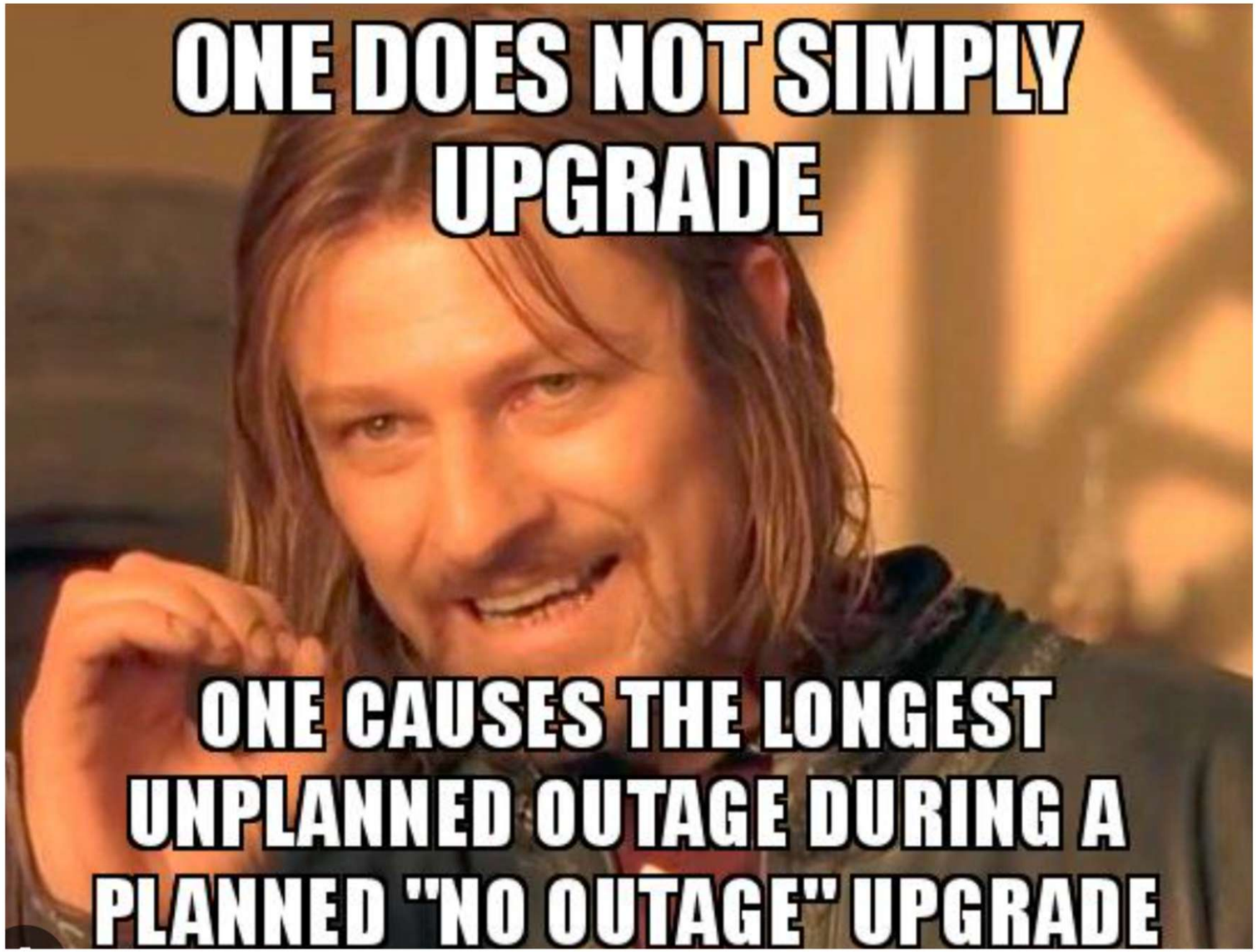


CrowdStrike

CrowdStrike

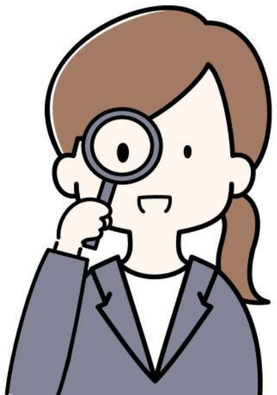
- **Date & Time**: July 19, 2024, 04:09 UTC
- **What**: Faulty configuration update to CrowdStrike Falcon Sensor software
- **Who**: ~8.5 million Windows PCs and servers worldwide
- **Scale**: One of the largest IT outages in history
- **Single Root Cause**: Logic error in "Channel File 291" configuration update

CrowdStrike

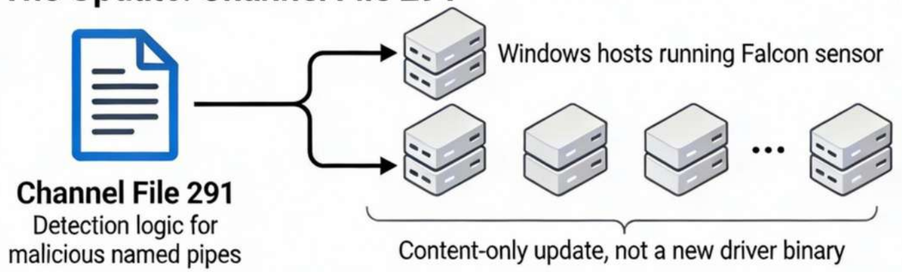


Root Cause - The Technical Details

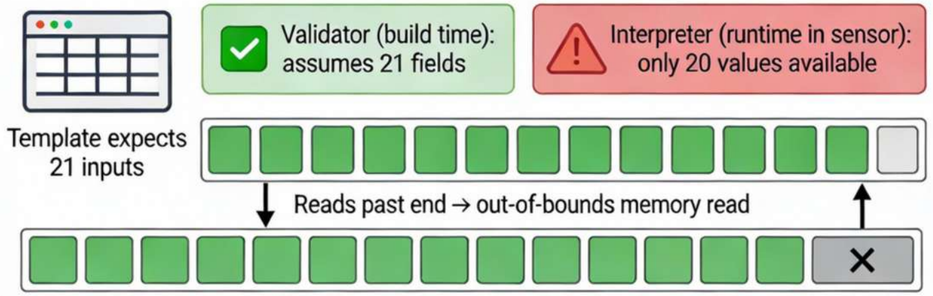
- **The Update**: "Channel File 291" - a configuration file for detecting malicious named pipes
- **The Bug**: Out-of-bounds memory read caused by parameter mismatch
- **The Mechanism**: Logic error in the file interacted with kernel-level code
- **Why Critical**: Falcon runs at the Windows kernel level (ring 0 - highest privilege)



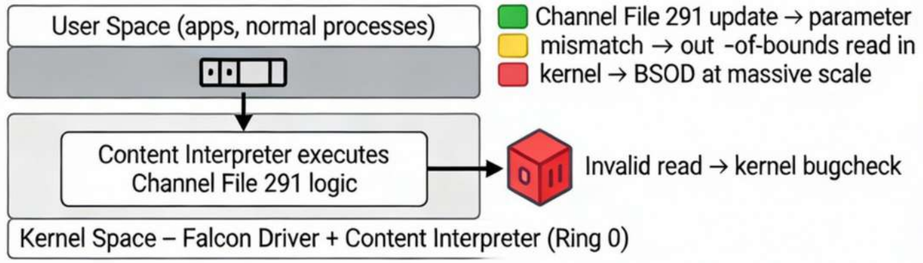
The Update: Channel File 291



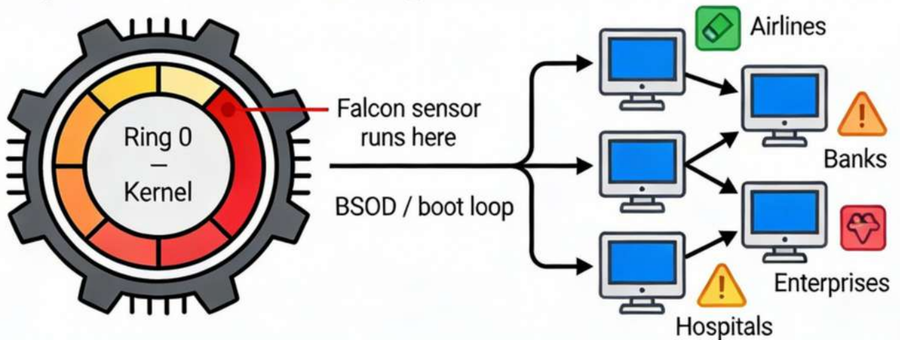
The Bug: Parameter Mismatch → Out-of-Bounds Read



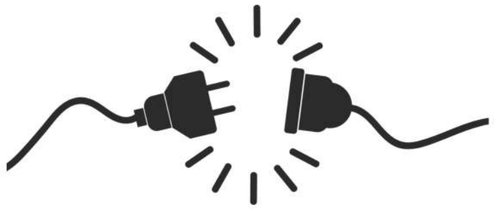
The Mechanism: Content Logic + Kernel-Level Code



Why Critical: Kernel Privilege → Global BSOD



The Cascade Effect - What Broke



- **Airlines**: AI, Indigo, Spicejet, Delta, Southwest airlines.
- **Hospitals**: Emergency departments unable to access patient records
- **Banks**: ATMs offline, transactions blocked
- **Government**: Courts, agencies unable to function
- **Media & Broadcasting**: TV stations went dark
- **Businesses**: Microsoft 365, Gmail, Slack all appeared down due to client devices

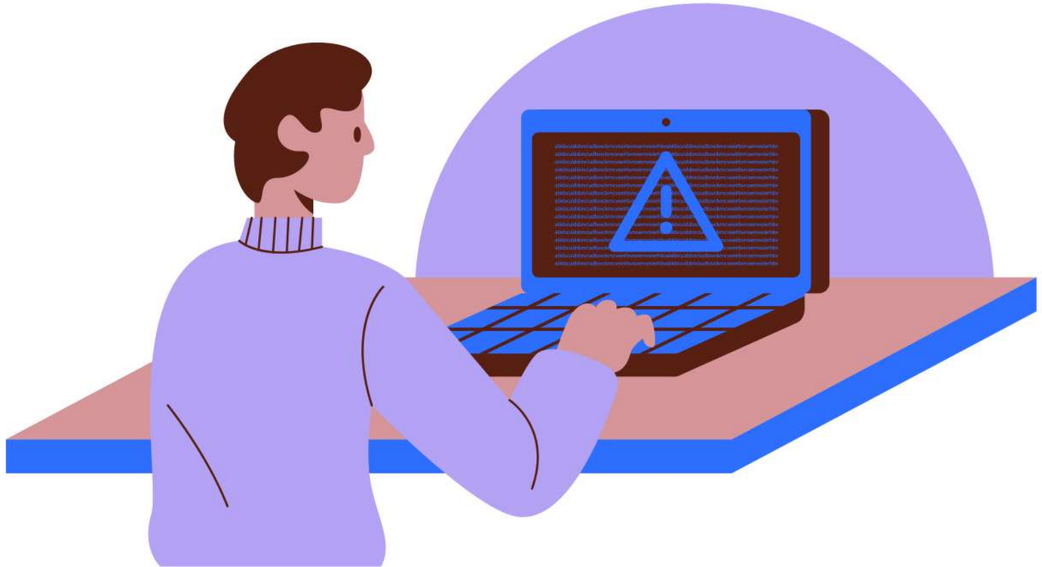
Employees when there is power cut or internet issues in middle of work

Key Lessons Learned

- Configuration is as dangerous as code
- Kernel-level access = zero margin for error
- Manual recovery at scale is a nightmare
- Rollout velocity matters for blast radius

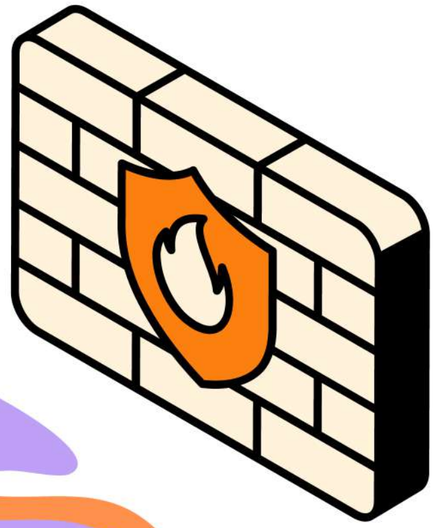
How to Prevent Similar Incidents

- Treat configuration with code-level rigor
- Implement multi-stage progressive rollouts



How to Prevent Similar Incidents

- Comprehensive validation testing
  - negative test cases: malformed data, null values
  - Chaos testing
- Build safety mechanisms into critical components
  - Safe mode boot options
  - Circuit breakers
- Maintain rollback mechanisms




aws
Google Cloud
CLOUDFLARE

Incident Overview - The Basics

Internal server error

Error code 500

Visit cloudflare.com for more information.

2025-11-18 11:35:32 UTC

You	London	[redacted]
Browser	Cloudflare	Host
Working	Error	Working

What happened?

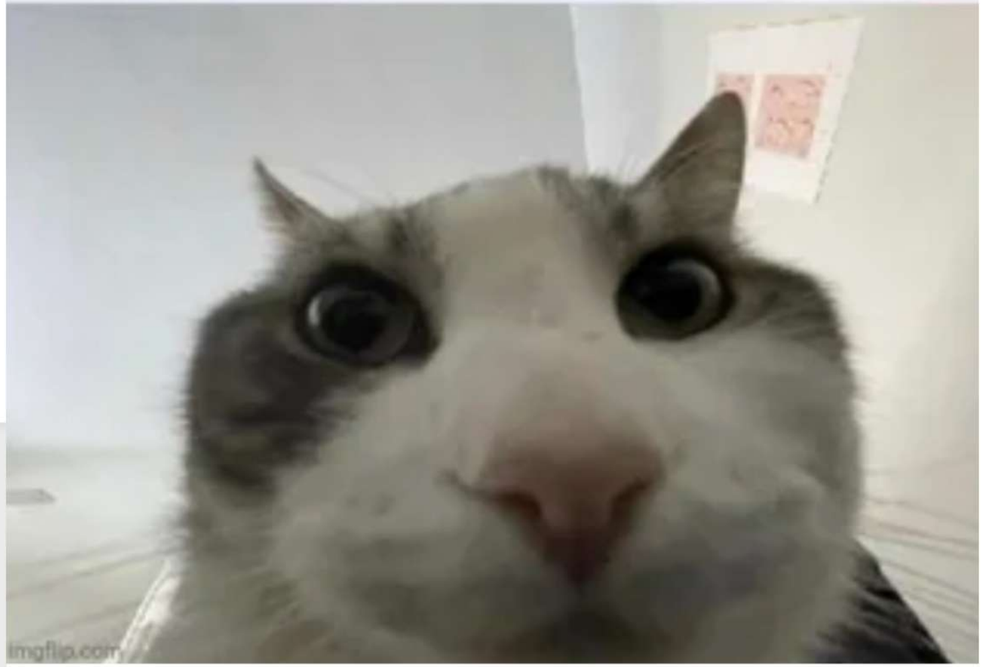
There is an internal server error on Cloudflare's network.

What can I do?

Please try again in a few minutes.

Cloudflare Ray ID: **9a072a3bce88467f** • Your IP: Click to reveal • Performance & security by Cloudflare

> Cloudflare failure
>
> You want read some details about this malfunction
>
> Portal with article is not working because it is using Cloudflare



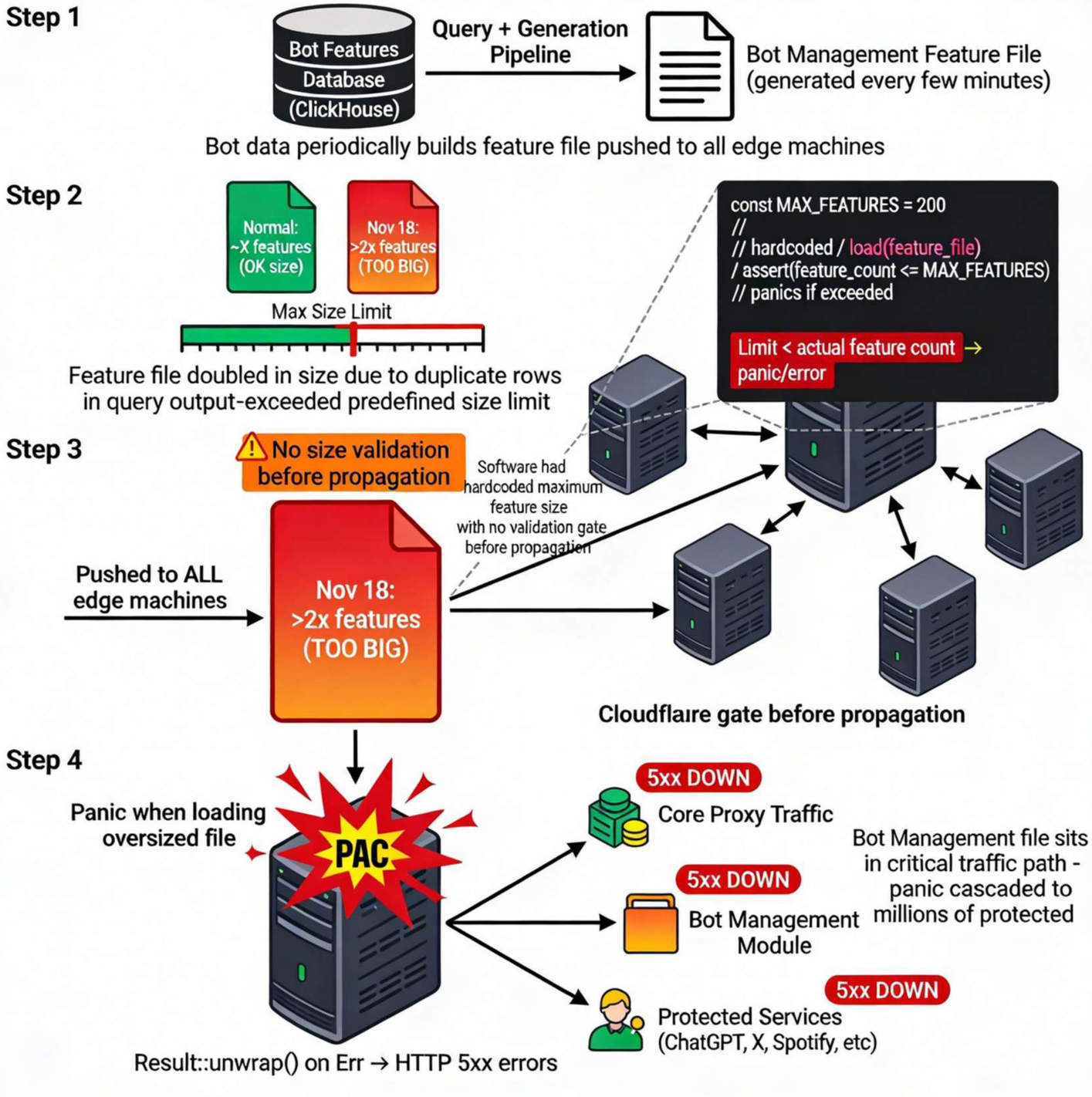
CloudFlare

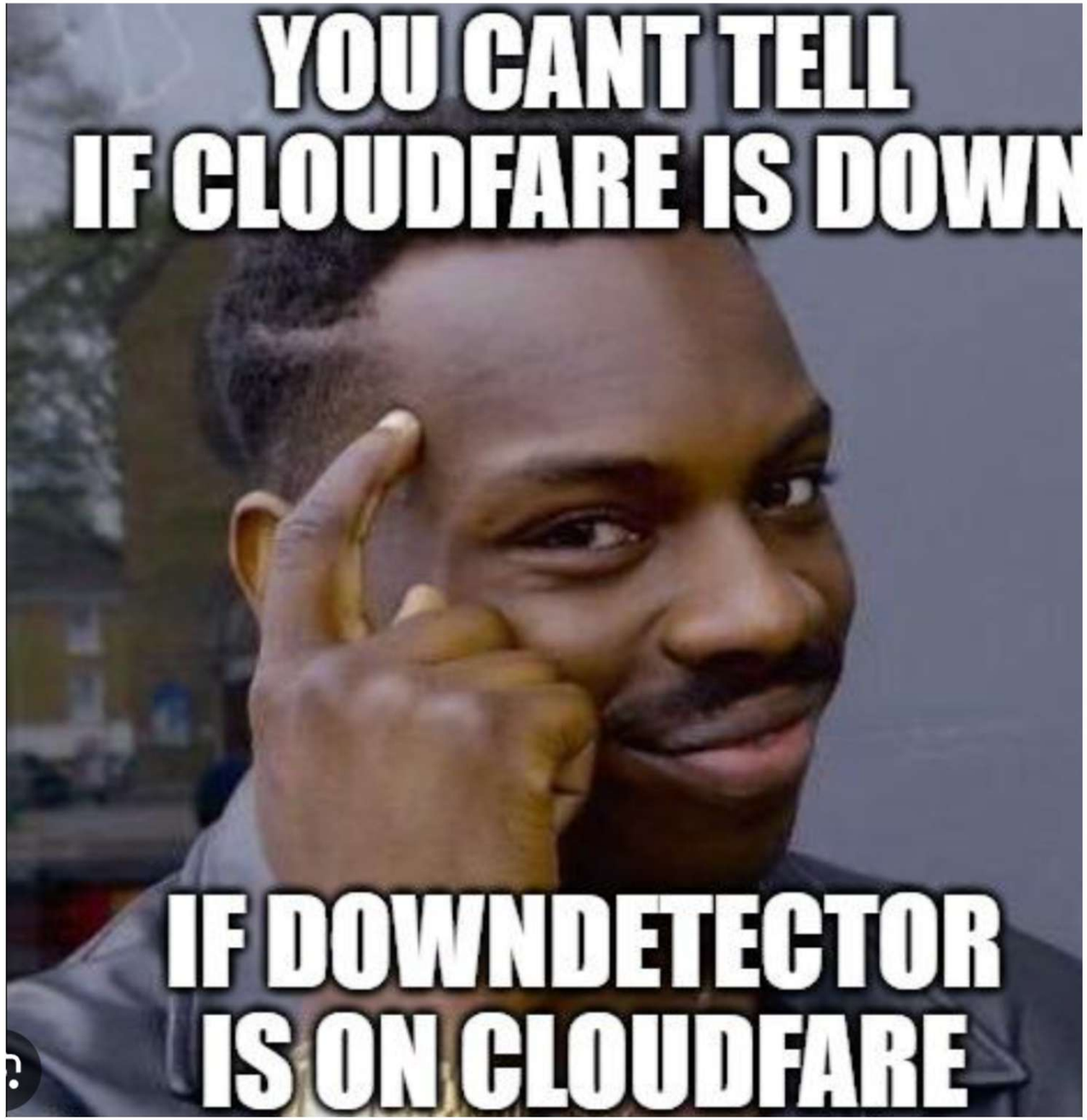
The Incidents - Timeline Overview

- **November 18, 2025 at 11:20 UTC:** Bot Management feature file bug causes cascading failures
- **Duration**: ~5 hours 38 minutes (11:20 UTC → 17:06 UTC)
- **December 5, 2025 at 08:47 UTC**: Configuration change for security mitigation causes brief outage
- **Duration**: ~25 minutes (08:47 UTC → 09:12 UTC)
- **Key Context**: Both incidents involved deployment propagation across a global edge network
- **Scale**: Millions of websites affected, 28% HTTP traffic, Zoom/LinkedIn/Discord down

November 18 Incident - Root Cause

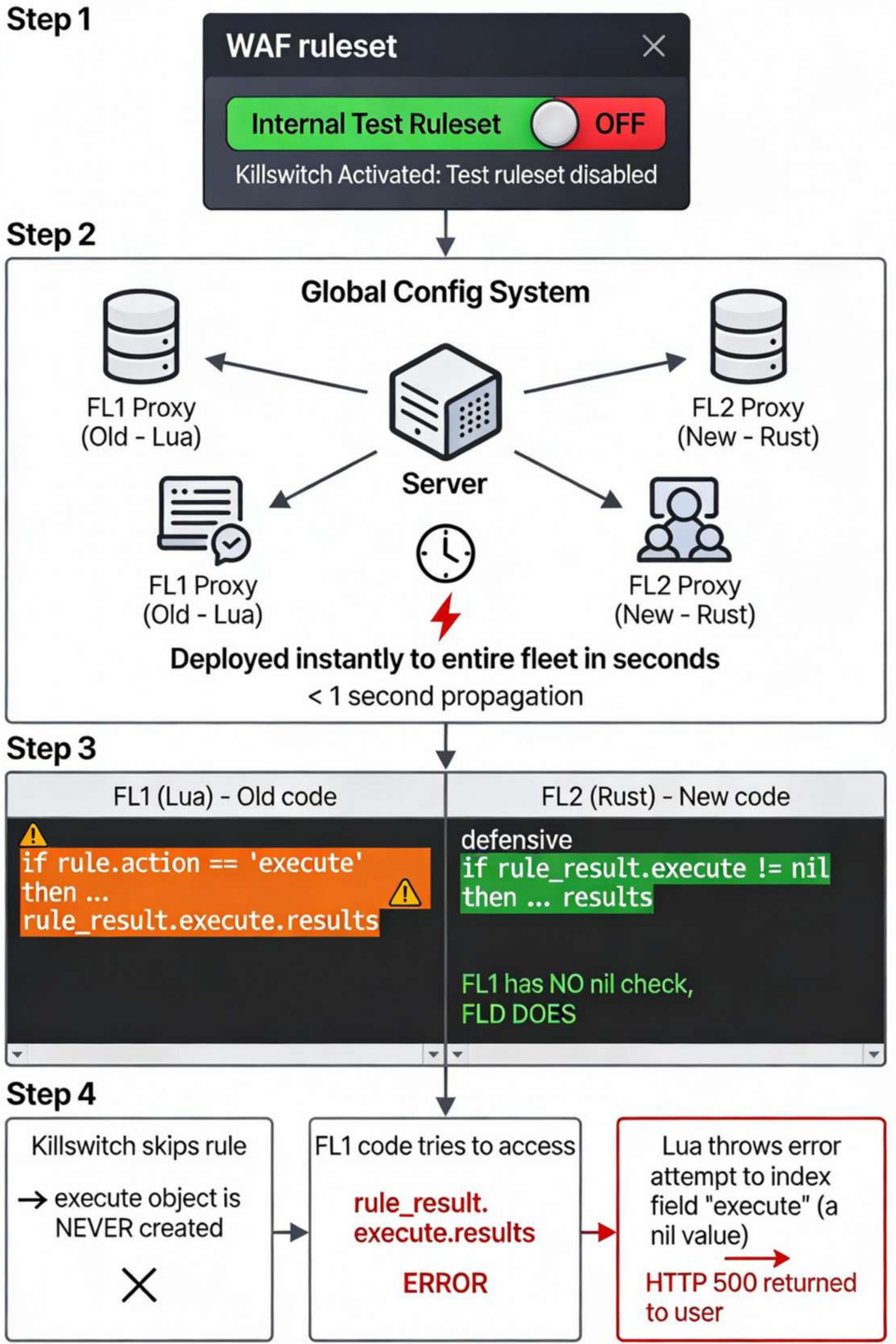
- **What Failed**: Bot Management feature file generation logic
- **The Bug**: Feature file exceeded predefined size limits in downstream systems
- **Root Cause**: Software had a hardcoded size limit that was below the actual file size; no validation before propagation
- **Why It Mattered**: The oversized file triggered failures across many Cloudflare services
- **Pattern**: Similar to CrowdStrike; a parameter mismatch between two components
- **Propagation**: The file propagated to the entire network before being detected


Step 1
Bot Features Database (ClickHouse)
Query + Generation Pipeline
Bot Management Feature File (generated every few minutes)
Bot data periodically builds feature file pushed to all edge machines
Step 2
Normal: ~X features (OK size)
Nov 18: >2x features (TOO BIG)
Max Size Limit
Feature file doubled in size due to duplicate rows in query output-exceeded predefined size limit
const MAX_FEATURES = 200
//
// hardcoded / load(feature_file)
/ assert(feature_count <= MAX_FEATURES)
// panics if exceeded
Limit < actual feature count → panic/error
Step 3
No size validation before propagation
Software had hardcoded maximum feature size with no validation gate before propagation
Pushed to ALL edge machines
Nov 18: >2x features (TOO BIG)
Cloudflare gate before propagation
Step 4
Panic when loading oversized file
PAC
5xx DOWN
Core Proxy Traffic
5xx DOWN
Bot Management Module
5xx DOWN
Protected Services (ChatGPT, X, Spotify, etc)
Bot Management file sits in critical traffic path - panic cascaded to millions of protected
Result::unwrap() on Err → HTTP 5xx errors


YOU CANT TELL
IF CLOUDFARE IS DOWN
IF DOWNDETECTOR
IS ON CLOUDFARE

December 5 Incident - Security Patch Gone Wrong

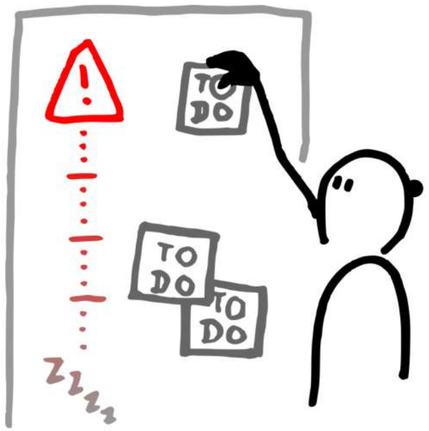
- **Context**: Recent industry-wide React Server Components vulnerability discovered
- **Config change**: Disabled a WAF test ruleset using a global "killswitch"
- **Applied globally**: Pushed to entire fleet in seconds
- **Old FL1 code**: Had a hidden bug; assumed "execute" object would always exist
- **The bug**: Tried to read from non-existent object → Lua error → HTTP 500
- **The Impact**: Caused errors for ~28% of HTTP traffic
- **Resolution**: Reverted within 25 minutes once detected


Step 1
WAF ruleset
Internal Test Ruleset
OFF
Killswitch Activated: Test ruleset disabled
Step 2
Global Config System
FL1 Proxy
(Old - Lua)
FL2 Proxy
(New - Rust)
Server
FL1 Proxy
(Old - Lua)
FL2 Proxy
(New - Rust)
Deployed instantly to entire fleet in seconds
< 1 second propagation
Step 3
FL1 (Lua) - Old code
FL2 (Rust) - New code
if rule.action == 'execute'
then ...
rule_result.execute.results
defensive
if rule_result.execute != nil
then ... results
FL1 has NO nil check,
FLD DOES
Step 4
Killswitch skips rule
→ execute object is NEVER created
FL1 code tries to access
rule_result.
execute.results
ERROR
Lua throws error
attempt to index
field "execute" (a
nil value)
HTTP 500 returned
to user

Services Impacted - Global & India Context

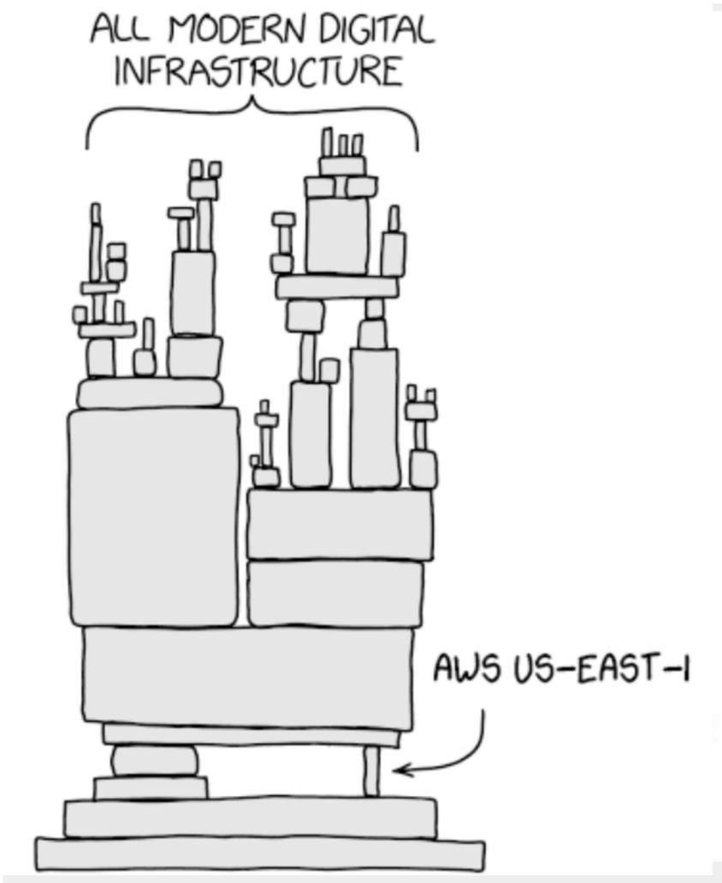
- **Globally Affected**: X (Twitter), ChatGPT, Spotify, Canva, Discord, LinkedIn, Zoom, Multiple platforms like- WhatsApp Web, banking portals, Paytm, Netflix, major news sites
- **Financial Services**: Online banking, payment platforms experienced degradation
- **E-commerce**: Amazon, Flipkart, and other retail sites behind Cloudflare CDN
- **User Experience**: Users saw 5xx errors, timeouts, or very slow page loads; not complete unavailability for most

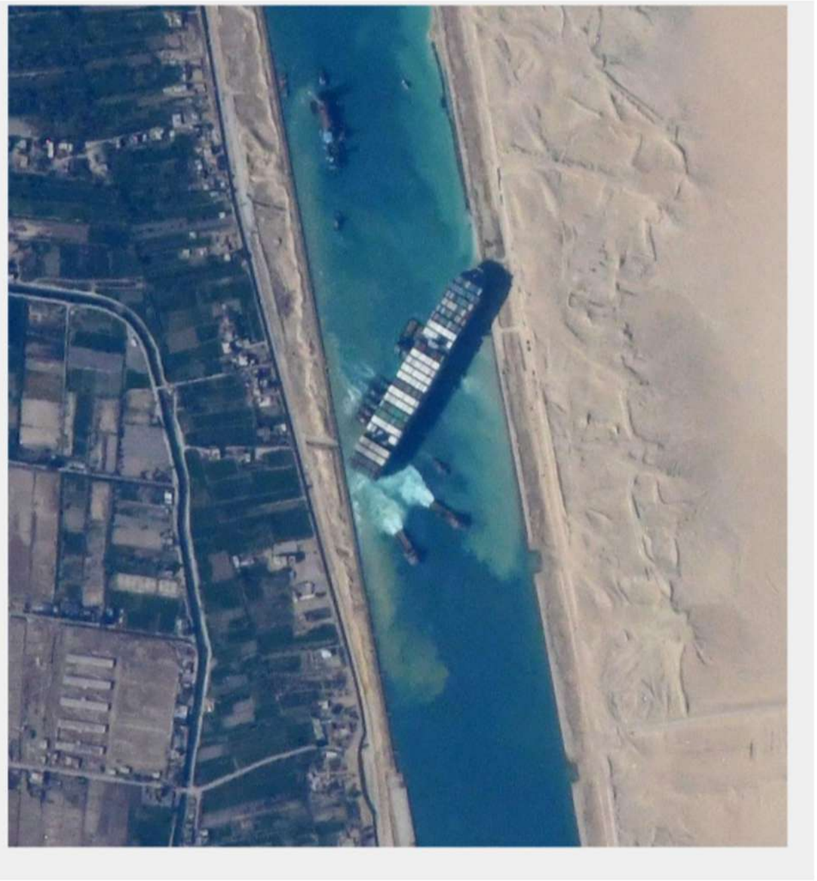
How to Prevent Similar Incidents



- **Enhanced pre-deployment validation**
  - Build production-equivalent staging environments for network changes
  - Implement "shadow mode" testing where changes run in parallel without affecting traffic
- **Gradual rollout for infrastructure changes**
  - Start with least-critical regions or edge locations
  - Automated traffic analysis comparing pre/post change behavior
  - Implement "single region" → "region pair" → "full deployment" progression
- **Fail-open**
  - Default to safe state instead of crashing when config is corrupt
  - Instead of HTTP 500, you might get degraded service. That's better than complete outage.

Incident Overview - The Basics





AWS US-EAST-1 (Oct 20' 25)

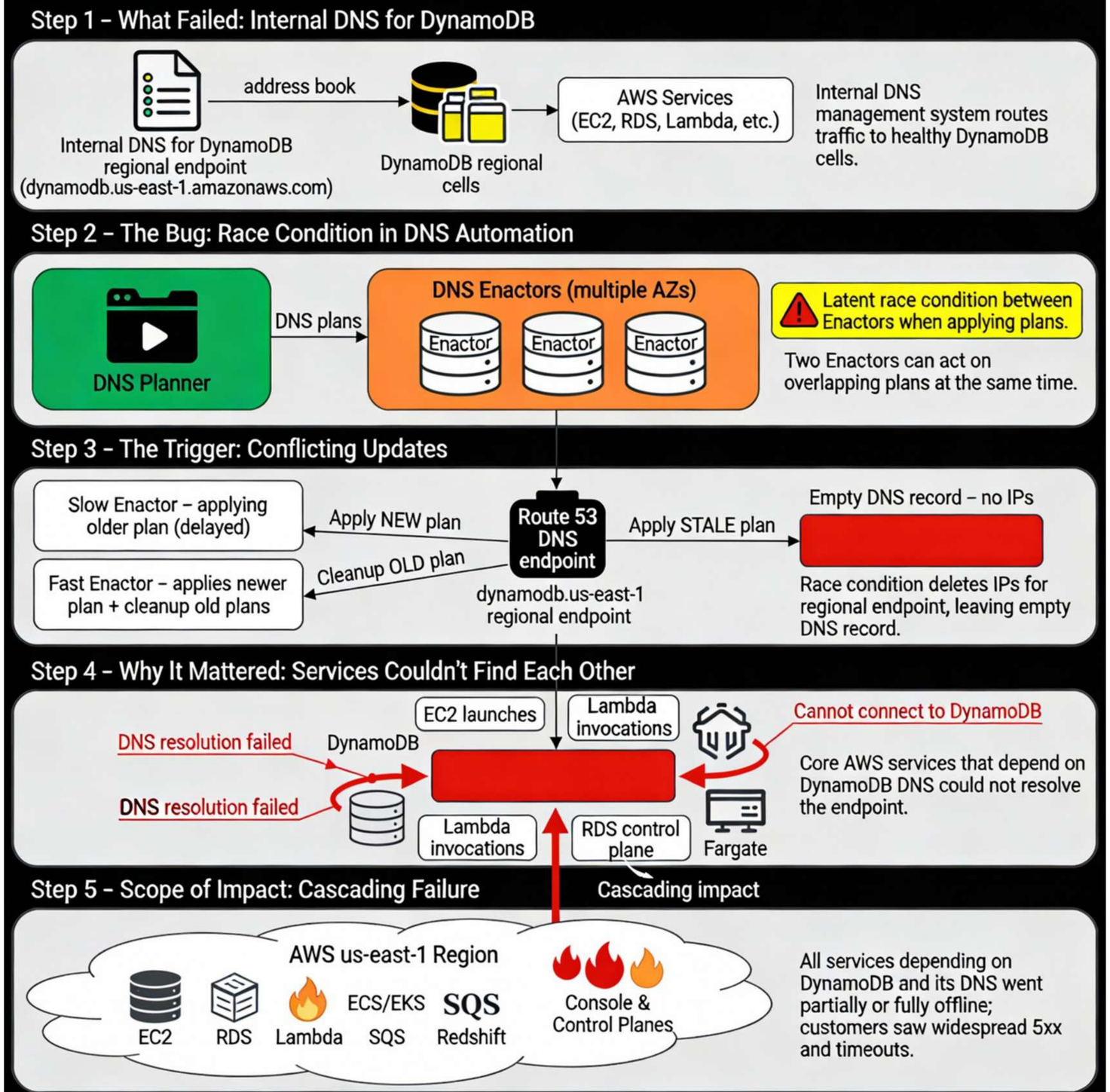
The Incident - Planetary Scale Disruption

- **Date & Time**: October 19, 2025, 11:49 PM PDT (October 20, 06:49 UTC)
- **Duration**: 14-15 hours of cascading failures
- **Region**: AWS US-EAST-1 (Northern Virginia)—single largest cloud region globally
- **Scope**: 140+ AWS services impacted across compute, storage, databases, networking
- **Root Cause**: Race condition in AWS internal DNS management system
- **Affected Applications**: Snapchat, Reddit, Disney+, Canva, and thousands more


AWS OUTAGE
BASED ON OUR INVESTIGATION THE ISSUE APPEARS TO BE RELATED TO
DNS
imgflip.com

Root Cause - DNS Race Condition in Database Infrastructure

- **What Failed**: AWS internal DNS management system for RDS and database services
- **The Bug**: Race condition in automated DNS update logic between two independent system components
- **The Trigger**: Two AWS components updated DNS records simultaneously; one deleted the critical regional endpoint entry
- **Why It Mattered**: AWS services couldn't find each other; cascading failure across EC2, RDS, Lambda, etc.
- **Scope of Impact:** All services depending on AWS database infrastructure went offline
- **Detection**: Took manual intervention; AWS automated systems couldn't self-heal

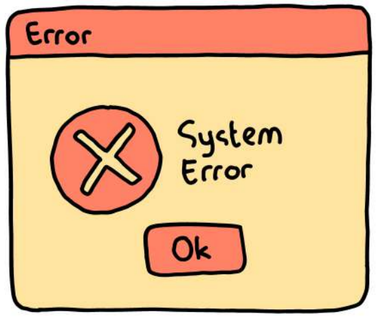

Step 1 – What Failed: Internal DNS for DynamoDB
address book
Internal DNS for DynamoDB regional endpoint (dynamodb.us-east-1.amazonaws.com)
DynamoDB regional cells
AWS Services (EC2, RDS, Lambda, etc.)
Internal DNS management system routes traffic to healthy DynamoDB cells.
Step 2 – The Bug: Race Condition in DNS Automation
DNS Planner
DNS plans
DNS Enactors (multiple AZs)
Enactor
Enactor
Enactor
Latent race condition between Enactors when applying plans.
Two Enactors can act on overlapping plans at the same time.
Step 3 – The Trigger: Conflicting Updates
Slow Enactor – applying older plan (delayed)
Fast Enactor – applies newer plan + cleanup old plans
Apply NEW plan
Cleanup OLD plan
Route 53 DNS endpoint
dynamodb.us-east-1 regional endpoint
Apply STALE plan
Empty DNS record – no IPs
Race condition deletes IPs for regional endpoint, leaving empty DNS record.
Step 4 – Why It Mattered: Services Couldn't Find Each Other
EC2 launches
Lambda invocations
DNS resolution failed
DynamoDB
DNS resolution failed
Cannot connect to DynamoDB
Core AWS services that depend on DynamoDB DNS could not resolve the endpoint.
Lambda invocations
RDS control plane
Fargate
Step 5 – Scope of Impact: Cascading Failure
Cascading impact
AWS us-east-1 Region
EC2
RDS
Lambda
ECS/EKS
SQS
SQS
Redshift
Console & Control Planes
All services depending on DynamoDB and its DNS went partially or fully offline; customers saw widespread 5xx and timeouts.

AWS down

Half of the internet:

The Cascading Collapse

Chain Reaction Explained :



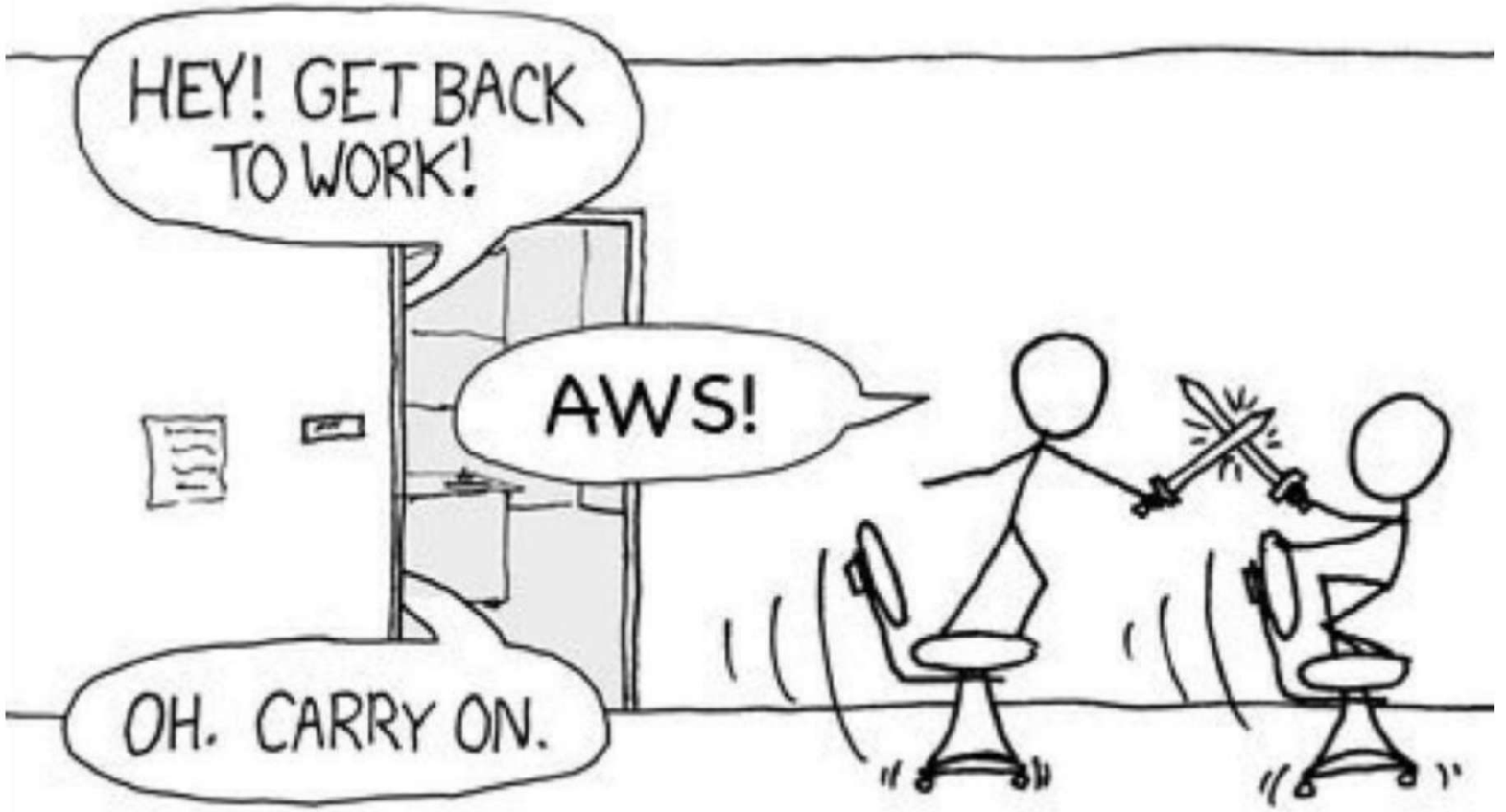
- DNS stops working for DynamoDB
- Other services can’t find DynamoDB
- Control plane components (like IAM, EC2 workflows) failed
- Network components start flagging healthy servers as unhealthy
- Even AWS engineers have trouble accessing internal tools
- Widespread service unavailability

How It was fixed?

- Engineers restored the correct DNS records so services could find DynamoDB again
- They stopped faulty automation that had caused the bad DNS update
- They throttled some operations (like instance launches) to reduce pressure on recovering systems
- Services gradually came back online over many hours, with full restoration by late afternoon the next day

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"AWS OUTAGE"



How to Prevent Similar Incidents:

- **True multi-region architecture**
  - Deploy critical services actively in multiple regions; Eliminate SPOF
- **Eliminate hidden dependencies**

How to Prevent Similar Incidents:

- **Design for cloud provider failure**
  - Implement multi-cloud or hybrid cloud for critical workloads
  - Build control planes that don't depend on cloud provider control planes
- **Independent observability and communication**
  - Status pages hosted outside your primary cloud provider
  - Alerting systems that don't depend on the infrastructure they're monitoring

Demo


Scan me!

The Checklist - Audit Your System Today

- **Configuration Safety**: Do configs get the similar reviews as code?
- **Urgency Process**: When urgent, do you enforce safety MORE, not less?
- **Amplification Awareness**: For each critical change, what's the blast radius?
- **Recovery Testing**: If system crashes, can you fix it remotely?
- **Dependency Mapping**: Chaos test your "independent" systems

Questions?

Let's Connect!

Unnati Mishra

Scan me!

Demo

Scan me!


Thank
you