Gmail

Marcus Birkenkrahe <birkenkrahe@lyon.edu>

The biggest-ever global outage: lessons for software engineers

The Pragmatic Engineer <pragmaticengineer+deepdives@substack.com> Tue, Jul 23, 2024 at 10:20 AM
Reply-To: The Pragmatic Engineer
<reply+2fgzth&217lk1&&bd3e9ffe89b0a7d5fb316a6c844be578d3efec714dbd34550d847bdfb321a7dc@mg1.substack.com>
To: birkenkrahe@lyon.edu

👋 Hi, this is Gergely with the monthly, free issue of the Pragmatic Engineer Newsletter. In every issue, I cover challenges at Big Tech and startups through the lens of engineering managers and senior engineers.

If you're not a full subscriber, you missed the issue on AI tooling reality check for software engineers, AWS shifting its focus away from infra, the Trimodal nature of tech compensation revisited, and more. Many subscribers expense this newsletter to their learning and development budget. If you have such a budget, here's an email you could send to your manager.

The biggest-ever global outage: lessons for software engineers

Cybersecurity vendor CrowdStrike shipped a routine rule definition change to all customers, and chaos followed as 8.5M machines crashed, worldwide. There are plenty of learnings for developers.

GERGELY OROSZ
JUL 23

READ IN APP

Unless you were under a rock since last week, you likely heard about the CrowdStrike / Windows outage that took down critical services like airlines, banks, supermarkets, police departments, hospitals, TV channels, and more, around the world. Businesses saw their Windows machines crash with the "Blue Screen of Death," and no obvious fixes – at least not initially. The incident was unusual in size and scale, and also because it involved software running at the kernel-level; a factor which gives this us all the more reason to take a look at it.

Today, we cover:

1. **Recap**. 8.5M Windows machines impacted across industries
2. **Root cause.** An update to naming rules for finding malicious processes somehow resulted in the *CSAgent.sys* process, which attempted to write to an invalid memory address, and crashed the operating system
3. **A very slow, manual fix.** Four days after the outage, recovery was ongoing, as every single impacted machine and host had to be fixed manually
4. **Who's responsible?** Obviously, CrowdStrike is and it's tempting to think Microsoft should share blame. A regulation from 2009 could also have played a role
5. **Learnings for software engineers.** Quantify potential impact, do canarying/staged rollouts, treat configuration like code, and more

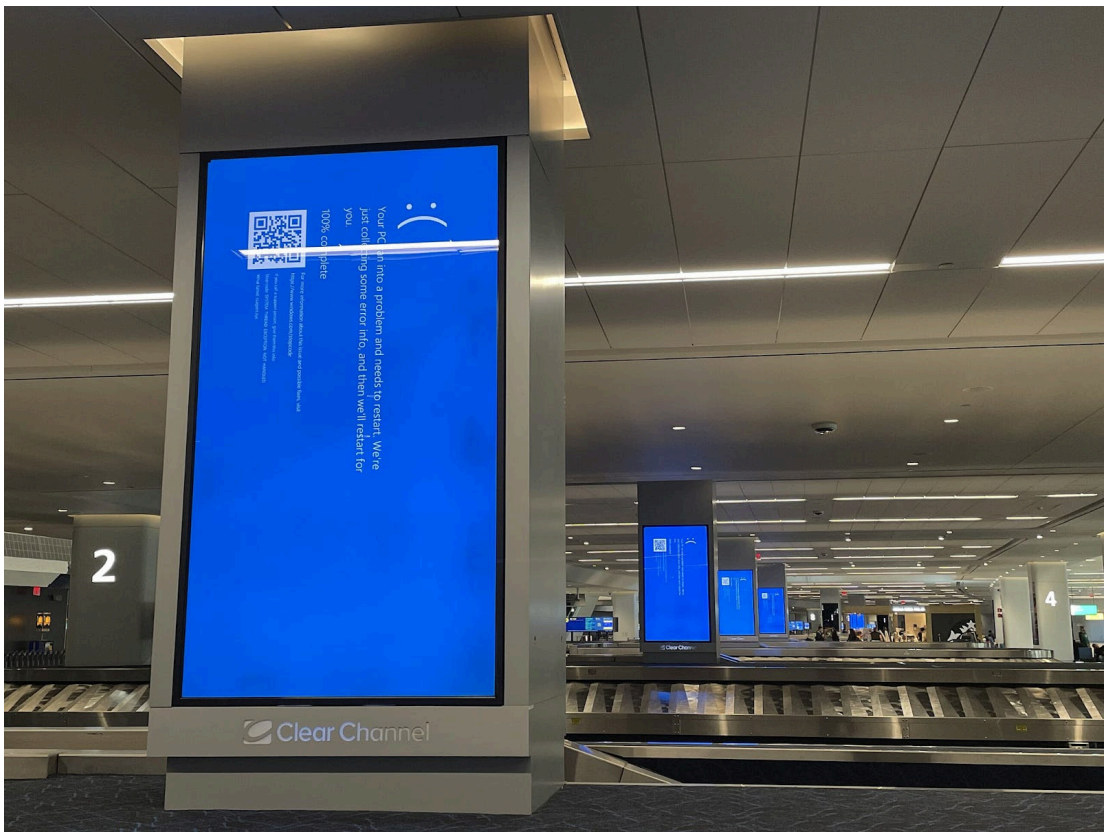
Note: this is the last issue before The Pragmatic Engineer goes on summer break. There will be no The Pulse on Thursday, and no new issues next week. We return on Tuesday, 6 August. Thanks for your continuing support of this publication!

1. Recap

Last Friday (19 July,) the largest-ever software-initiated global outage hit machines worldwide. Millions of Windows 10 and 11 operating systems used by societally-critical businesses like airlines, banks, supermarkets, police departments, hospitals, TV channels, etc, suddenly crashed with the dreaded

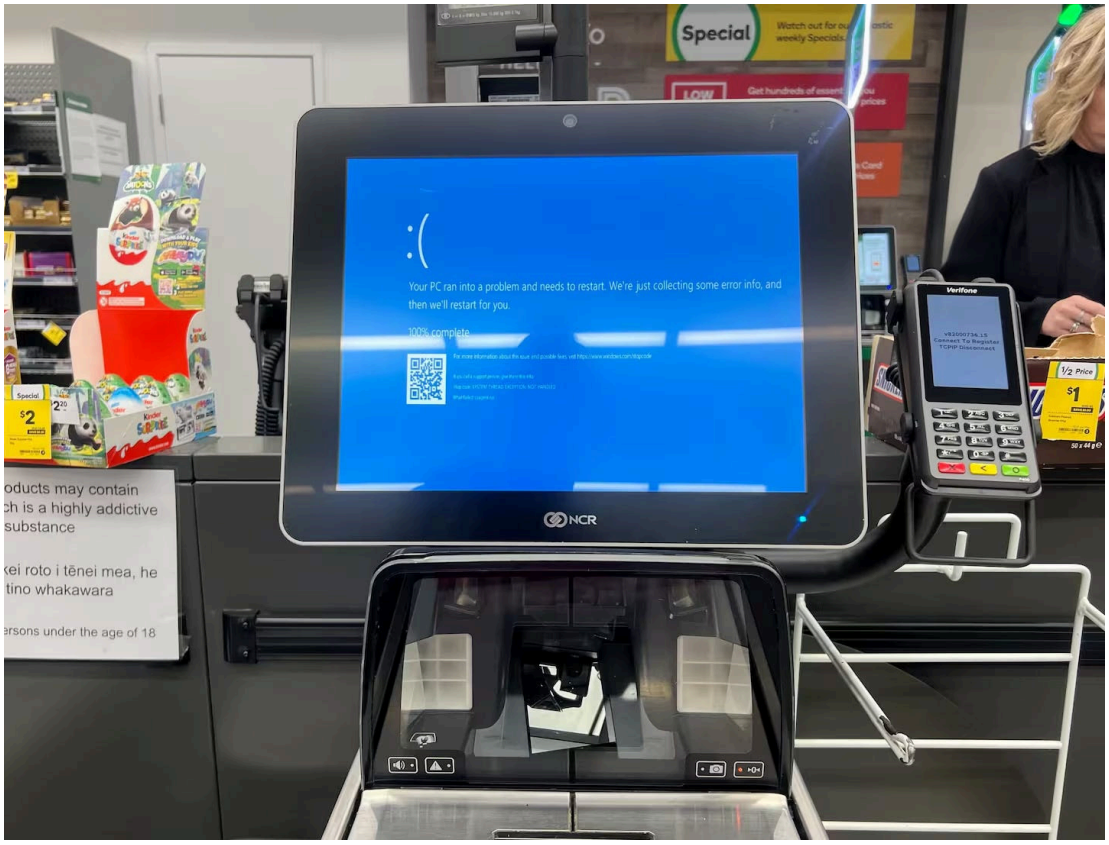
“Blue Screen of Death,” and no obvious way to fix them. This was a truly global outage; the US, Europe, Asia, and Australia, were all hit.

Global air travel descended into chaos, and in Alaska the emergency services number stopped working. In the UK, Sky News TV was unable to broadcast, and McDonalds had to close some of its Japanese outlets due to cash registers going down. In total, tens of thousands of businesses and millions of people were impacted. Meanwhile, in the world of Formula One racing, the Mercedes team saw its computers crash at the Hungarian grand prix. Ironically, one of the team’s sponsors is… CrowdStrike. Some photos of the outage in the wild:

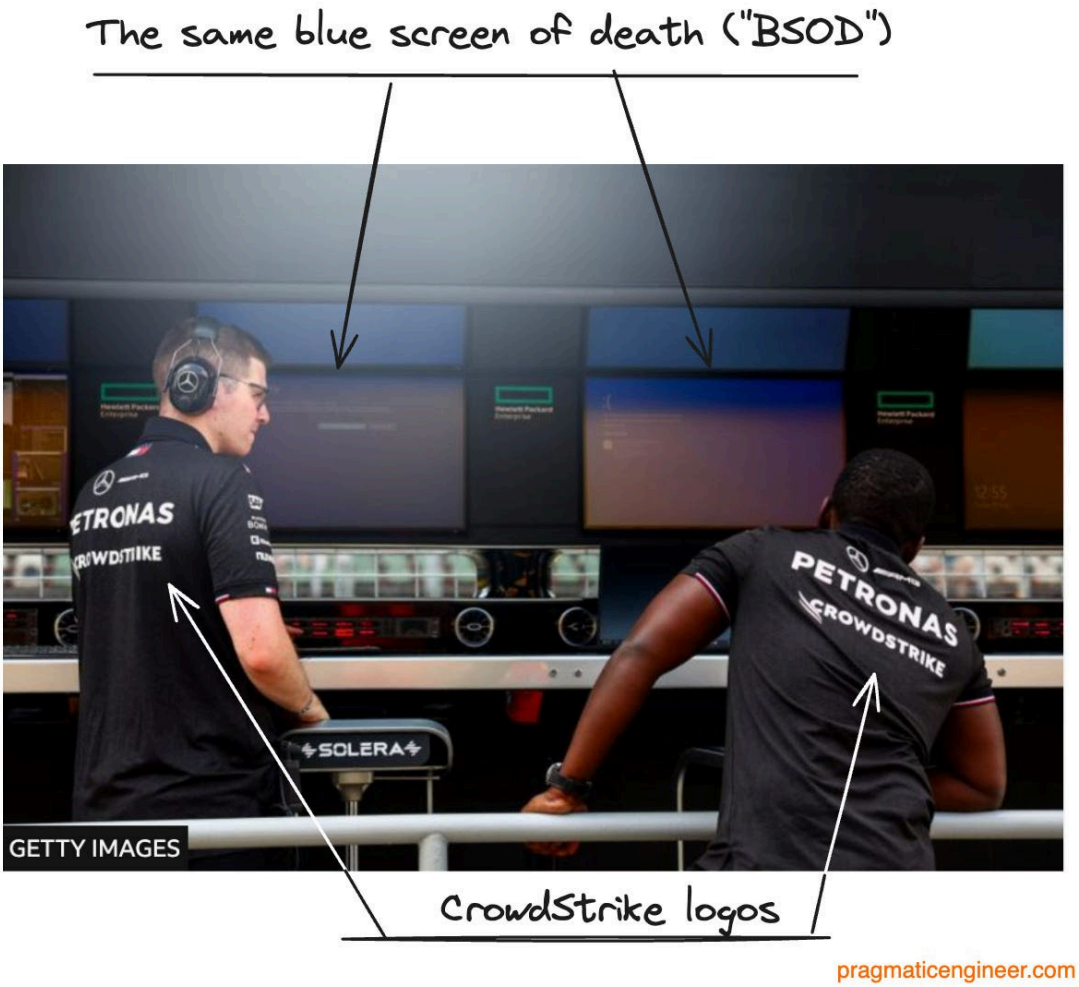


Conveyor belt screens at LaGuardia airport, New York, on 19 July 2024. Source: Wikipedia

Disneyland Paris was impacted, where staff switched to paper printouts to display wait times for rides. Source: The Points Guy



Self-service checkout in a supermarket in New Zealand (Auckland.) Source: The New Zealand Herald



The Windows crash caused by the CrowdStrike update caused issues for the F1 team sponsored by CrowdStrike. Source: BBC / Getty

All the business victims of this mega crash were customers of cybersecurity company CrowdStrike, which is the market leader in “endpoint security,” with around 20% market share. It installs software on Windows / Linux / Mac machines, and runs antivirus, firewalls, intrusion detection and prevention systems (IDP,) among others. What unleashed the global carnage was a single update by Crowdstrike to its ‘Falcon’ product.

We know 8.5M Windows machines were impacted globally from Microsoft sharing this number, later confirmed by CrowdStrike. Worst-hit of all might be Delta airlines, where around a third of flights (5,000) were canceled in three days. Even on day 4, Delta had to cancel another 1,000 flights as it recovered, and is on the hook for cash refunds for impacted customers.

2. Root cause

A few hours after Windows machines running CrowdStrike's software started crashed, the company issued an update:

> "CrowdStrike is actively assisting customers affected by a defect in a recent content update for Windows hosts. Mac and Linux hosts were not impacted. The issue has been identified and isolated, and a fix has been deployed. This was not a cyberattack."

What happened is the company pushed out a "content" file (a binary file) to all customers at once, which then crashed the operating system. But how did it happen? As the incident was ongoing, some devs attempted to reconstruct what happened. Here are details from Patrick Wardle:

1. The process that crashed Windows is called "CSAgent.sys"

2. The instruction that crashed is the Assembly instruction "mov r9d, [r8]." This instructs to move the bytes in the r9d address to the r8 one. The problem is that r8 is an unmapped address (invalid), and so the process crashes!

```
mov     rax, [rdx+8]
mov     r8, [rax+r11*8] ; R11: 0x14
                        ; RAX: buffer w/ pointers (though at 0x14 addr is foo'barred)
                        ;
                        ; R8: unmapped invalid memory addr (e.g. 0xffff9c8e`0000008a)
jnz     short loc_1400E14E8 ; (likely) take
test    r8, r8
jz      short loc_1400E14F4
movzx   r9d, word ptr [r8]
jmp     short loc_1400E14F0
--------------------------------------------------------------------------------

                        ; CODE XREF: sub_1400E11D0+30B↑j
test    r8, r8          ; check R8 != NULL
jz      short loc_1400E14F4 ; don't take
mov     r9d, [r8]       ; Faulting Instruction: 0xffff9c8e`0000008a is not paged in, so
```

The culprit: assembly instructions that crashed Windows machines across the globe. Source: Patrick Wardle on X

3. The crash was caused by the CSAgent.sys process reading a new "content" file CrowdStrike pushed to all clients called "C-00000291-*.sys" (where * can have additional characters.) Something went wrong related to this file and the parsing of it.

A day later, CrowdStrike shared more details:

1. The goal of the update was to detect maliciously-named pipes. CrowdStrike's Falcon product observes how processes communicate on a

machine, or across the network, to try and pinpoint malicious activity. The update was adding a new rule file to filter for suspiciously named pipes. A named pipe in the Windows world is a "named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients." These pipes can be used for inter-process communication (two processes talking to each other; here's an example of processes sensing files between one another, or to communicate over the network. *Named pipes are a common concept with operating systems for interprocess communication: Unix also uses this concept.*

2. Released a new configuration file with new rules/naming. CrowdStrike calls config files that define behavior rules, like names for suspicious names pipes, "Channel files." They store all these channel files in the location C:\Windows\System32\drivers\CrowdStrike\. These are numbered files, and the rules for named pipes are under number 291. Therefore, every file with the naming pattern "C-00000291-*.sys" is a rule for this category.

CrowdStrike released a new naming file in the update.

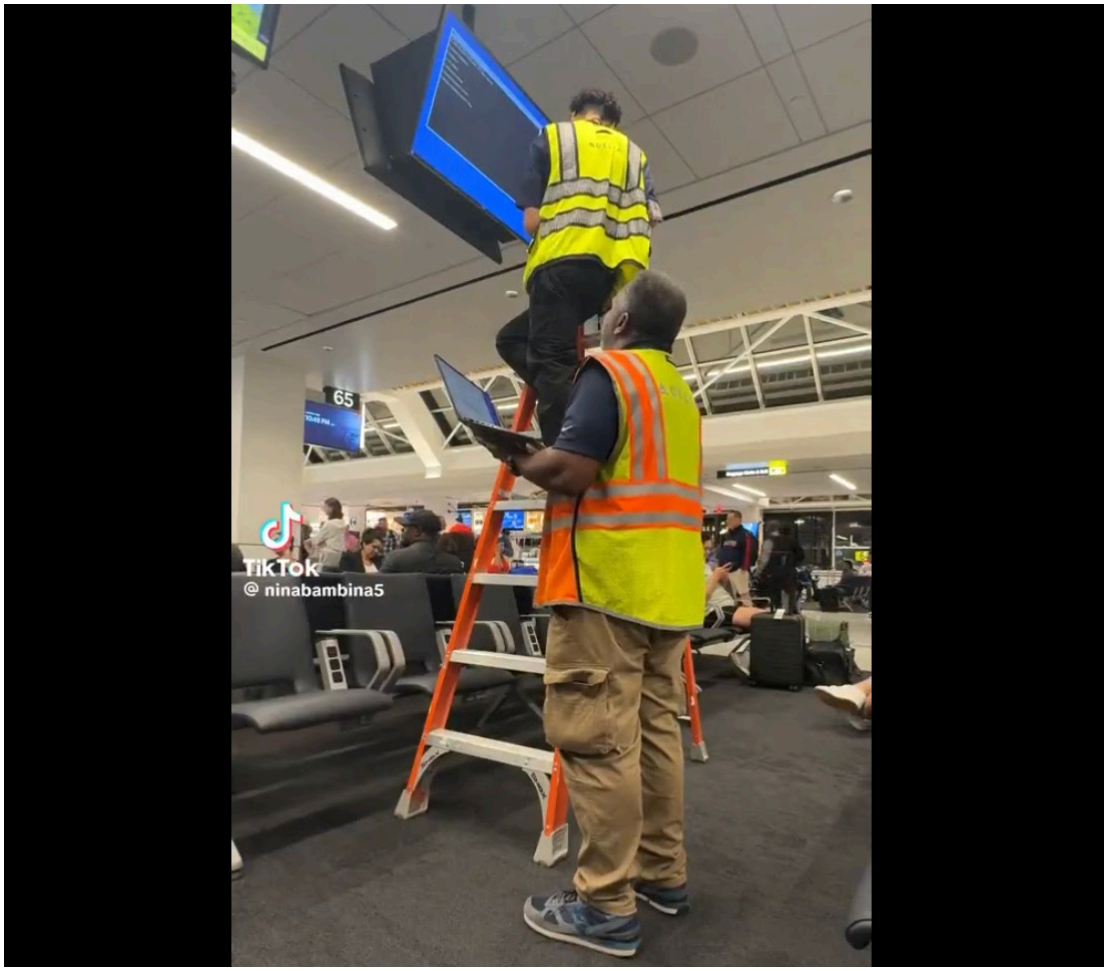
3. An unhandled error crashed the process and the operating system. While I'm itching to know about what the error actually was, CrowdStrike has only shared a very brief summary:

> "The configuration update triggered a logic error that resulted in an operating system crash. This is not related to null bytes contained within Channel File 291 or any other Channel File."

So, somehow, parsing these new naming rules resulted in an Assembly-level instruction that tries to move a memory location to an invalid location. This is what made Windows devices crash everywhere.

3. A slow, manual fix

Mitigating this outage was a lot more complicated than usual because a simple revert was insufficient. IT staff had to physically access each individual machine:



Fixing an impacted Windows machine because there was no remote fix option. Source: techAU on X

CrowdStrike posted mitigation steps for IT admins and developers wanting to get themselves unblocked, a few hours after the incident. The steps were:

1. Boot Windows into Safe Mode or the Windows Recovery Environment
2. Navigate to the C:\Windows\System32\drivers\CrowdStrike directory
3. Locate the file matching “C-00000291*.sys” and delete it
4. Boot the host

The recovery process might need a local administrator on a machine with the right to delete the offending file. The steps are specialized enough that regular users would struggle to perform the recovery: and so at most companies it’s up to IT staff to manually fix every machine. Plus, at many places all Windows laptops were impacted. An IT admin shared a glimpse of the task, posting an image of 120 of 2,000 laptops to be fixed in one weekend, ideally!

Some laptops to be reset. Source: Dunken K Bliths on X

As software engineers, when we see a highly manual process our first thought is whether we can automate it, or do it faster in a clever way. With 8.5M machines needing resets, it's obvious a manual process is incredibly time consuming. So independent developers, and also Microsoft, stepped in:

- iOS developer and Windows tinkerer Adam Demasi created the Unistrike tool a day later. With some additional setup, you can create a USB stick to plug into each impacted machine for faster recovery
- Microsoft also released a similar recovery tool the day after the outage
- Two days later, CrowdStrike shared that they were testing a new, faster recovery technique for customers

Four days after the outage, most of the 8.5M impacted Windows devices weren't fixed. It turns out that crashing operating systems at scale is a lot harder to recover at scale than applications, for which patches can be sent out to clients (mobile and desktop apps,) or when the fix can be done server side (services, backend applications, web apps.)

4. Who's responsible?

It was a little amusing that the news initially reported this as a "Microsoft outage" or a "Windows outage" because it's a bit distant from the facts. So who "owns" the world's biggest-ever software crash?

CrowdStrike – obviously

Needless to say, most blame lies with CrowdStrike. We can only speculate at this point about which areas were skipped, or done insufficiently thoroughly. Hopefully, we will learn more in a public-facing postmortem. Meantime, here are some questions that CrowdStrike should be asking – and, most likely, is:

1. Was the change tested, and how? Was the change in this config file (C-00000291-*.sys) tested in manual and automated scenarios? If so, how did the tests pass, and why did the crash happen in production? A more interesting question that only CrowdStrike can answer is how the configs were tested in an automated way; and indeed, were they? *We know testing environments can never replicate production in full, so it's expected that bugs can make their way through tests, undetected.*

2. Were these config changes dogfooded? Was this change rolled out to CrowdStrike staff, before release to the public? If yes, did some CrowdStrike employees also see their OSs crash? If yes, then why did the rollout proceed? If there was dogfooding, but no employees' machines crashed; an interesting question is: why not?

3. Was there a canary rollout? We cover the topic of canarying in Shipping to Production:

> "Canarying" comes from the phrase "canary in the coal mine." In the early 20th century, miners took caged canary birds with them underground. The bird has a lower tolerance for toxic gasses than humans do, so if the bird stopped chirping or fainted, it was a warning sign to miners that gas was present, and for them to evacuate.
>
> Today, canary testing means rolling out code changes to a smaller percentage of the user base, then monitoring the health signals of this deployment for signs that something's not right. A common way to implement canarying is to either route traffic to the new version of the

> code using a load balancer, or to deploy a new version of the code to a single node."

Canarying is a subset of staged rollouts:

> "Staged rollouts mean shipping changes step by step, evaluating the results at each stage before proceeding. Staged rollouts typically define the percentage of the user base to get the changed functionality, or the region where this functionality should roll out, or both.
>
> A staged rollout plan may look like this:
>
> - Phase 1: 10% rollout in New Zealand (a small market to validate changes)
> - Phase 2: 50% rollout in New Zealand
> - Phase 3: 100% rollout in New Zealand
> - Phase 4: 10% rollout, globally
> - Phase 5: 25% rollout, globally
> - Phase 6: 50% rollout, globally
> - Phase 7: 100% rollout, globally
>
> Between each rollout stage, a criteria is set for when the rollout can continue. This is typically defined as there being no unexpected regressions and the expected changes to business metrics being observed."

Did CrowdStrike use these approaches, or was it more of a "YOLO rollout," where the configuration file was pushed to all customers at the same time? Right now, we don't know.

From the incident response communication, it sounds like the change was more a "YOLO rollout" because the changed file was labeled as "content," not business logic. *This is despite it containing rules on how to detect named pipes, which you could argue is business logic that should be rolled out in phases, not all at once!*

4. Does CrowdStrike assume that binary ("content") files cannot break software running at kernel level? Common rollout strategies for shipping code were likely absent when shipping these new configuration files. Did CrowdStrike assume – implicitly or explicitly – that these "content" files could not crash the process?

CrowdStrike's software operates at the kernel level in Windows, meaning its process is operating with the highest level of privileges and access in the OS. This means it can crash the whole system; for example, by corrupting part of the OS's memory. CrowdStrike operating at this level is necessary for it to oversee processes running across the OS, and to discover threats and vulnerabilities. But this also means that an update – even an innocent-looking content file! – can cause a crash.

5. Did the company ignore a previous similar outage? A Hacker News commenter working at a civic tech lab shared that, a few months ago, CrowdStrike caused a similar outage for their Linux systems. This dev summarized:

> "Crowdstrike did this to our production linux fleet back on April 19th [2024], and I've been dying to rant about it.
>
> The short version is we're a civic tech lab, so have a bunch of different production websites made at different times on different infrastructure. We run Crowdstrike provided by our enterprise. Crowdstrike pushed an update on a Friday evening that was incompatible with the up-to-date Debian stable. So we patched Debian as usual, everything was fine for a week, and then all of our servers across multiple websites and cloud hosts simultaneously hard crashed and refused to boot.
>
> When we connected one of the disks to a new machine and checked the logs, Crowdstrike looked like a culprit, so we manually deleted it, the machine booted, tried reinstalling it and the machine immediately crashed again. OK, let's file a support ticket and get an engineer on the line.
>
> Crowdstrike took a day to respond, and then asked for a bunch more proof (beyond the above) that it was their fault. They acknowledged the bug a day later, and weeks later had a root cause analysis that they didn't cover our scenario (Debian stable running version n-1, I think, which is a

> supported configuration) in their test matrix. In our own post mortem there was no real ability to prevent the same thing from happening again: "we push software to your machines any time we want, whether or not it's urgent, without testing it," seems to be core to the model."

These details suggest that CrowdStrike could or should have been aware that it can – and does – crash kernel processes with updates. If so, the obvious question is why this outage did not serve as a warning to tweak the rollout process, as opposed to just improving testing?

In fairness, a company like CrowdStrike has hundreds of engineering teams, and one team observing an outage is information that will not necessarily spread through the organization. Still, the CrowdStrike process crashing the OS was surely a known vulnerability, as it's the most obvious way to brick a customer's machine which it is meant to defend.

Microsoft / Windows?

Why can CrowdStrike run processes at kernel level which can crash an operating system? After all, Apple made changes to MacOS to run third-party software at user level, not kernel. From Electric Light, in 2021:

> "For some years now, Apple has been encouraging third-party developers to move away from kernel extensions to equivalents which run at a user level rather than in Ring 1. However, it has only been in the last year or so that Apple has provided sufficient support for this to be feasible. Coupled with the fact that M1 Macs have to be run at a reduced level of security to be able to load third-party kernel extensions, almost all software and hardware which used to rely on kernel extensions should now be switching to Apple's new alternatives such as system extensions. This article explains the differences these make to the user."

So on Mac, the same CrowdStrike process would run in the user space, and if it crashes it would not take the whole system down with it.

However, on Windows and Linux, antivirus and other cybersecurity software usually runs at the kernel level, and always has done. So why hasn't Microsoft followed Apple's approach and banned third parties from the kernel

space? Turns out that a Symantec complaint in the 2000s, and EU regulation, played a role.

Regulation to blame?

The Wall Street Journal asked Microsoft why it won't limit third-party software like CrowdStrike to run only in the user space, not the kernel space. Its response:

> "A Microsoft spokesman said it cannot legally wall off its operating system in the same way Apple does because of an understanding it reached with the European Commission following a complaint. In 2009, Microsoft agreed it would give makers of security software the same level of access to Windows that Microsoft gets."

Ironically, all of this started in 2006 with Microsoft wanting to make its kernel more secure for Windows Vista. From CIO.com at the time (emphasis mine):

> "Security vendors like Symantec are in a state of heightened sensitivity these days as they've begun to compete with Microsoft head-on, and the specter of further antitrust actions looms over Microsoft's every move in the security space. Last week, the European Union's spokesman on competition, Jonathan Todd, warned that the market could be threatened if Microsoft doesn't allow security vendors a fair chance of competing.
>
> Symantec and other security vendors dislike PatchGuard because it prevents them from accessing the Windows kernel. They say it will stop them from delivering important features like Symantec's "anti-tampering" technology, which prevents malicious programs from modifying Symantec's own software.
>
> PatchGuard will also make it more difficult for security vendors to protect against malicious software that takes advantage of kernel-level bugs, said Eric Sites, vice president of research and development with Sunbelt Software. (...)
>
> Microsoft declined to be interviewed for this article, but in an interview with IDG News last week a Microsoft executive said that PatchGuard was simply an effort to prevent the kernel from being misused.

> "We think that there's a significant amount of confusion around… certain security features in the product that we think raise the foundation," said Stephen Toulouse, a senior product manager in the Security Technology Unit. **"What we're doing is we're walling off the kernel from attackers, because the functionality that is currently there was never meant to be used by anybody — by software vendors or attackers."**

In the end, Symantec and other vendors won. Microsoft could only "ban" security vendors from running in the kernel space if it also did not run its own security software there. So while Microsoft could be *seen* as partly responsible for this crash, the company had little choice in the actions which created the circumstances for it to happen!

There would have likely been a way, though: if Microsoft moved their own security solution – such as Windows Defender – out of the kernel space, closing it off to *all* security vendors, including itself. Doing so would likely mean a large enough re-architecture of the Windows security stack. It would also limit the capabilities of third-party vendor solutions, and any such change would trigger outcry and more complaints to regulators by security vendors. It would be no different to the complaints and escalations of 2006, when Vista attempted to lock vendors out of the kernel space.

5. Learnings for software engineers

Here are some learnings that us software engineers can take from this incident, as things stand:

Quantify the impact of software crashing *everywhere*

What happens if your company's product crashes irrecoverably for a couple of hours? Ignore the fact that this seems so unlikely as to be impossible – because it has just happened to CrowdStrike. If it happened, what would the impact be on your company and the outside world? For example:

- If Amazon crashed worldwide for a few hours, sellers would lose revenue and a subset of shoppers could lose access to essential items. Amazon would lose revenue and suffer reputational damage.
- If TikTok crashed for hours worldwide for a few hours, brands would not be able to serve ads, and customers would feel indifferent, slightly

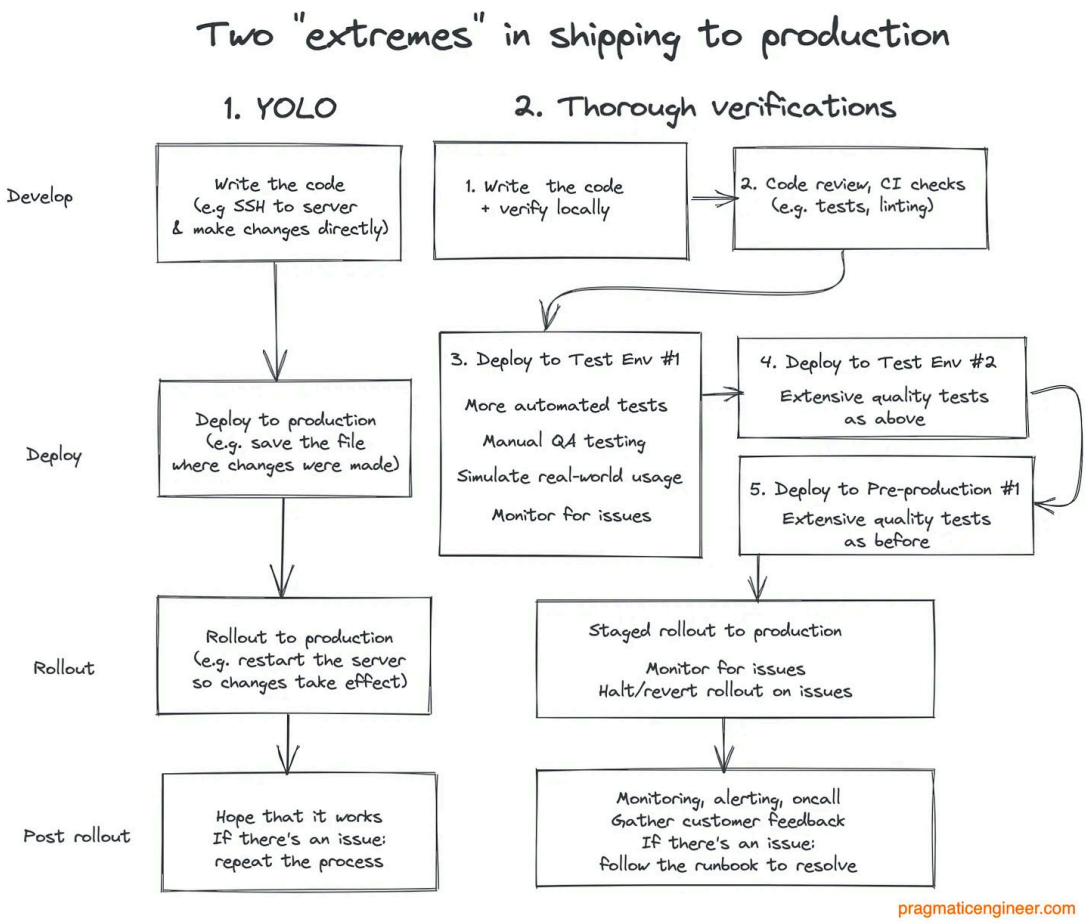
annoyed, or furious about not being able to use the social platform. Far-fetched theories could emerge about TikTok being blocked, the company would lose ad revenue, and users would temporarily flock to alternatives like Instagram Reels and Snap.

- If a major phone and internet carrier crashed, the impact would be far worse than the two above, combined. Businesses would struggle to operate and emergency services could be impacted. The damage would be reputational, lasting, and government intervention could also follow. Last November, we covered what happened when Half of Australia's internet was cut off for 14 hours

This exercise is helpful because it can give a sense of how expensive an outage could be. Knowing the "blast radius" can help get buy-in to make systems more resilient, and make it quicker to detect and mitigate incidents.

Review how things reach production

What needs to happen for a code or asset change to be shipped to all customers? We go deep on this topic in Shipping to Production. As a recap, here are two extremes in shipping to production: CrowdStrike seems to have chosen the "YOLO" option for this change, and it cost them dearly:



Two "extremes" of shipping to production: YOLO and thorough verification. Source: Shipping to Production

Do canarying / staged rollouts

If your software crashing *everywhere* has a big enough "blast radius" to make a failure unacceptable, then do not roll out changes to all customers at once! Do a canary or a staged rollout.

It's true that canarying and staged rollouts are overkill for certain products, like those with few users, or which do not generate revenue, or are experimental. Setting up canarying or staged rollouts is effort and does slow down rollout. But if your product is used by many people, or is critical enough, then this rollout strategy is non-negotiable. Take it from former Google Chrome engineer Marc-Antoine Ruel:

> "From the get go, Google Chrome had 3 release channels plus nightly builds:
>
> canary (nightly), dev, beta, stable. Each increases the blast radius.

> Dev did get broken builds. Beta broke in subtle ways. This release scheme reduced the blast radius.
>
> Net result: 126 stable releases!”

Treat configuration like code

Staff engineer Jacques Bernier formerly worked at Twitch and has shared how Amazon treated code changes:

> “The Amazon build and deploy system is one that I miss deeply. It was so powerful. And it considered all changes equal. It included code, dependencies all the way down to the operating system and infrastructure in the same build and deploy pipeline with multiple stages.
>
> Config change is code. Infrastructure change is code change. Dependency update is code change. Code change is code change. It is all the same.
>
> Staged rollout is one of the best ways (if not the best) to reduce blast radius.”

What changes do dependencies/vendors “silently” push?

CrowdStrike broke most customers’ businesses because it silently and automatically shipped business logic changes. Even if customers wanted to “block” a change, or only allow it for a subset of machines at first, they could not.

It’s a good reminder that software can be broken not just by code, but by your dependencies or vendors. So now is a good time to consider these questions:

- How are dependencies (libraries, frameworks) updated? Are they done automatically, or manually? This is especially relevant when working with package managers that might get these automatically.
- What about vendor dependencies – SDKs or APIs? Are you the one making changes, or are vendors pushing silent changes?

List all the points which could be affected by a faulty “silent” change from a third-party you use and (currently) trust.

An outage is no one person's fault

It's easy to blame whoever wrote an offending piece of code for a crash; perhaps an intern lacking experience, or a veteran engineer having a bad day. But pointing the finger of blame at individuals is the wrong approach. Microsoft Veteran Scott Hanselman summarizes why a failure at this scale is never one person's fault (emphasis mine:)

> "Here's the thing folks. I've been coding for 32 years. When something like this happens it's an organizational failure. Yes, some human wrote a bad line. Someone can "git blame" and point to a human, and it's awful.
>
> But it's the testing, the CI/CD, the A/B testing, the metered rollouts, an "oh s**t" button to roll it back, the code coverage, the static analysis tools, the code reviews, the organizational health, and on and on.
>
> It's always one line of code but it's NEVER one person. Implying inclusion policies caused a bug is simplistic, reductive, and racist. Engineering is a team sport. Inclusion makes for good teams. Good engineering practices make for good software. Engineering practices failed to find a bug multiple times, regardless of the seniority of the human who checked that code in.
>
> **Solving the larger system thinking SDLC matters more than the null pointer check.** This isn't a "git gud C++ is hard" issue and it damn well isn't an DEI one."

Takeaways

Widespread outages are always bad, but one upside is that they force us engineers to pause and reflect:

- Can something similarly catastrophic happen at my company, and if so, how?
- What would the impact be in my situation?
- What do we do to avoid being the "next CrowdStrike?"

There's no better time than now to make a case to your leadership for investing properly in reliability. The CrowdStrike outage is now officially the largest-ever software outage on the planet, and customers have suffered

heavy financial and reputational damage. The financial loss is still unclear for CrowdStrike, but you can assume it will be huge, as some businesses will seek compensation for the damage done.

For CrowdStrike, the reputational damage could hardly be worse. Until a few days ago, the company was the gold standard in endpoint security compliance. No longer: its name is linked with the biggest outage anybody's seen. After such a high-profile blunder that reveals the company had no staged rollout processes in place for business rule changes ("channel files,") the reputation of Crowdstrike has suffered a hit which it will take a long time to recover from.

No business wants such a blow from *a single* bad deploy, but it's happened. If you see gaps in your company's release processes – testing, rollout, monitoring, alerting, etc – then now is the time to take your concerns and suggestions to the table! Talk with your manager or skip-level; they will be more likely to champion ideas which make production systems resilient.

CrowdStrike is certain to learn its lesson, and doubtless its future release processes will be world class. *Good luck to the team there (and teams at all affected customers) for mitigating the outage, and for work ahead at CrowdStrike to overhaul internal processes.*

Let's hope many companies follow suit, so this historic event ends up being a net positive learning experience for the tech industry.

We've previously covered outages with interesting learnings. Check out these for more analysis and learnings, and for making systems more reliable:

- Three cloud providers, three outages: three different responses
- AWS's us-east-1 outage
- Inside Datadog's $5M outage
- Half of Australia knocked offline for 14 hours
- Inside the longest Atlassian outage
- Four failed attempts to determine why Roblox was down for three days

---

You're on the free list for The Pragmatic Engineer. For the full experience, become a paying subscriber. Many readers expense this newsletter within their company's training/learning/development budget.

Upgrade to paid

This post is public, so feel free to share and forward it.

Share The Pragmatic Engineer

If you enjoyed this post, you might enjoy my book, **The Software Engineer's Guidebook**. Here is what Tanya Reilly, senior principal engineer and author of The Staff Engineer's Path said about it:

> "From performance reviews to P95 latency, from team dynamics to testing, Gergely demystifies all aspects of a software career. This book is well named: **it really does feel like the missing guidebook for the whole industry.**"

Get The Software Engineer's Guidebook

LIKE COMMENT RESTACK