Episode Book

EPISODE #5848

The Libraries Nobody Thinks About

PUBLISHED	10 October 2026 at 20:18 UTC
DURATION	27:13
LISTEN	myweirdprompts.com/episode/unsung-software-libraries-38087/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.3
SOURCES	16 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Daniel asked us to celebrate the software nobody thinks about: the small, fundamental pieces of logic that run everywhere and get replaced by no one. We start with zlib, still shipping thirty-one years of continuous maintenance by Jean-loup Gailly and Mark Adler, and with SQLite's own claim of over a trillion active databases. Then we get to the uncomfortable half: OpenSSL securing two-thirds of the encrypted web on about two thousand dollars a year in donations, and the multi-year xz backdoor, where a patient attacker spent years pressuring a solo maintainer before planting malicious code in a release. It's a conversation about permanence as a product — and about who pays for it.

TRANSCRIPT



Corn

Okay. Before we get anywhere near this one, I need to say what I actually thought when I read it.



Herman

Go on.



Corn

My first reaction was delight, and my second reaction was that this is a trap. Daniel has asked us to talk about the software nobody thinks about, which is a lovely idea, and it is also an invitation to spend thirty minutes being smug about things we happen to know. So I want to try and not do that.



Herman

I'll be the judge of whether you succeed.



Corn

You'll be the one failing alongside me. Daniel's framing, in his words, is that we celebrate operating systems and languages and applications and now AI models, but underneath all of it are much smaller pieces of logic that perform fundamental tasks so reliably nobody gives them a second thought. He uses a thermostat as his entry point. A basic one is just a threshold, heat on below this, off above it. The slightly more serious version is PID control, proportional, integral, derivative, three terms, and variations of it run industrial automation and robotics and a great deal else. His actual question is: what are the software equivalents? Named algorithms and named implementations, reused across thousands of projects, unchanged for decades, and so essential that replacing them would be brutal. Who wrote them, how far did they spread, and who is keeping them alive now?



Herman

That last clause is where the episode is.



Corn

It is. Because the honest answer to "who's keeping them alive" is going to be uncomfortable. So let's start with the one that is probably on your phone right now, several hundred times over.



Herman

zlib. Before anything else, zlib. And I want to be careful about how I say this, because the temptation is to present it as archaeology, and it isn't. zlib 1.3.2 came out on the seventeenth of February this year. This is a live library. The copyright line in the readme reads nineteen ninety-five to twenty twenty-six, Jean-loup Gailly and Mark Adler. That's not a relic they've dusted off for an anniversary. That's thirty-one years of continuous maintenance by the same two names.



Corn

One of whom did compression and the other decompression, which is a division of labour I find weirdly moving.



Herman

It's exactly the right division, actually, and it's the reason the thing is solid. Gailly wrote the compressor, Adler wrote the decompressor, and then they spent three decades making sure the decompressor could read everything the compressor ever produced. That's the promise. Once you can read a stream, you can read it forever.



Corn

So walk it back. Where does the algorithm actually come from?



Herman

Deflate. The format was defined by PKWARE for PKZIP 1.93a in October of nineteen ninety-one. Then it shows up in the Info-ZIP tools, then GNU gzip, then zlib wraps it in a library with a stable interface. The formal spec for the wrapper is RFC 1950, May nineteen ninety-six, authored by L. Peter Deutsch and Gailly. And what Deflate actually does is two old ideas stacked. First half is LZ77-style back-referencing. You keep a sliding window of what you've already emitted, and when the next chunk of input matches something already in the window, instead of writing that chunk again you write a pointer: go back this many bytes, copy this many. The second half is Huffman coding. You build a variable-length



Herman (continued)

code table where frequently occurring symbols get short codes and rare ones get long ones.



Corn

Those two compound, which is the whole trick.



Herman

They compound hard, which is why a text file compresses to a third of its size in the time it takes to blink. And here's the thing to hold onto for the rest of the episode: Deflate is not the best compressor. It was never the best compressor. By the mid-nineties there were things that compressed smaller, and now there's zstd and lzma2 that will beat it on ratio and, in zstd's case, beat it on speed as well. Deflate is not the winner on merit. Deflate is the one that is everywhere.



Corn

Why?



Herman

Because it landed in the right place at the right time and then became the floor. It's a crucial component of Linux, of macOS, of iOS. It shipped inside the PlayStation 4. Every PNG file you've ever downloaded is a zlib stream with a header on it. Every HTTP response that arrives gzip-encoded is Deflate underneath. Every zip archive, every PDF with compressed streams, half the file formats you've never thought about. And once a format is baked into a file format that's baked into an operating system that's baked into a hundred million devices, the question stops being "is this the best option" and becomes "what would it cost to move."



Corn

And the answer is: more than anyone wants to pay.



Herman

There was a comment on Hacker News a few years back that put it better than I can. The algorithm is showing its age, RFC 1950 is twenty-five years old and Deflate goes back to thirty-three-year-old PKZIP, but we still use zlib nearly everywhere, not least because it just works, no matter what language or hardware you're on.



Corn

"No matter what language or hardware you're on" is doing an enormous amount of work in that sentence.



Herman

It's the whole case. There are zlib implementations in C, in Java, in Rust, in Go, in JavaScript, in Python, in every embedded toolchain anyone has ever shipped. They all produce byte-compatible output. You can compress on a mainframe from 1998 and decompress on a phone in your hand and the bytes match. You cannot casually replace that. You can add zstd next to it, and people have, but you cannot remove zlib, because somewhere in your stack is a thing that only speaks Deflate and will never be rewritten.



Corn

SQLite is the same shape of story, isn't it? Except the numbers are worse.



Herman

The numbers are absurd. SQLite's own site claims it is likely used more than all other database engines combined. Billions and billions of copies in the wild. And then they try to actually count, and the count comes out at over one trillion active SQLite databases.



Corn

Say the number again.



Herman

One trillion. Ten to the twelfth. And the arithmetic behind it is mundane, which is why I believe it. Four billion plus smartphones in active use, each one holding hundreds of SQLite files, because that's how your contacts are stored, your messages, your browser history, your app settings, your calendar, all of it. Add every Android device, every iPhone, every Mac, every Windows 10 and 11 install, every Firefox and Chrome and Safari, Skype, iTunes, Dropbox, TurboTax, QuickBooks, most TVs and set-top boxes, most automotive multimedia systems.



Corn

The car.



Herman

Your car has a SQLite database in it and it is probably the least interesting thing about your car, which is saying something given what's under the hood these days.



Corn

That page, the "Most Widely Deployed" page, when was it last touched?



Herman

April twenty-first of this year. Still making the claim. And the detail I love most on that page is the peer list. SQLite names its own neighbours in ubiquity. It names the original zlib implementation by Gailly and Adler. It names the original reference implementation of libpng. It names libjpeg from the Independent JPEG Group. And then it guesses, modestly, that SQLite is probably the second most widely deployed library in the world, after libz.



Corn

So SQLite's own assessment is that there is exactly one library more everywhere than SQLite, and it's the one we just spent ten minutes on.



Herman

And notice the company libpng and libjpeg are keeping. Every image you've ever seen on a screen went through one of those two. libpng is the reference implementation of the Portable Network Graphics format, libjpeg is the Independent JPEG Group's reference implementation of JPEG, and between them they've decoded essentially every photograph taken since the early nineties. Written by small groups. Maintained by small groups. Essentially unchanged at the format level for decades, because the format has to be unchanged, because your photos from 1996 have to still open.



Corn

That's the through-line, isn't it? The reason these things don't get replaced isn't that nobody could write a better one. It's that a format, once it's in the world, has to keep working. Permanence is the feature.



Herman

Permanence is the product. And that's what makes the second half of this conversation so uncomfortable, because if a piece of code has to keep working forever, then somebody has to keep working on it forever, and we have collectively decided not to pay for that.



Corn

So these things are everywhere and they have been everywhere for decades. The obvious question: who's keeping them alive?



Herman

The numbers are shocking. April 2014. Heartbleed. OpenSSL at that moment was securing around two-thirds of the world's websites. Two-thirds of the encrypted web ran on this one library. And it was being done by a group of volunteers with very little funding.



Corn

How little?



Herman

Steve Marquess, who was president of the OpenSSL Software Foundation, wrote at the time that OpenSSL typically received about two thousand dollars in donations a year and had one employee working full time on the open source code.



Corn

Two thousand dollars a year for two-thirds of the encrypted web.



Herman

That's the sentence. That's the whole episode, honestly. Two thousand dollars a year, one full-time person, and the lock icon on two out of every three sites you visited.



Corn

And Heartbleed wasn't a backdoor, right? It was a bug.



Herman

A buffer over-read in the heartbeat extension. You send a malformed heartbeat message claiming a longer payload than you actually sent, and the server reads past the end of the buffer and hands you back whatever was in memory. Which could be session keys, passwords, private key material. Anybody could ask, and the server would tell you a little more than it meant to. It sat there for about two years before anybody found it.



Corn

And then the response.



Herman

The Linux Foundation stood up the Core Infrastructure Initiative, raised roughly three million dollars, funded a security audit and two full-time developers. Which was the right thing to do and also, in retrospect, a patch on a structural problem rather than a fix for it. Marquess himself said the thing that should be quoted more often. He said OpenSSL does, quote, belong to the people, but that it is neither realistic nor appropriate to rely on



Herman (continued)

volunteers for critical infrastructure.



Corn

"Belong to the people" is a lovely phrase and it's exactly the problem. If it belongs to everyone, it belongs to no one, and no one is on the hook for the invoice.



Herman

Which is why the xz backdoor is the story that should keep people up at night. Because somebody read that sentence and treated it as an opportunity.



Corn

Walk us through it, because it's a multi-year thing and I want people to feel the duration of it.



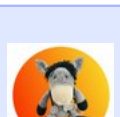
Herman

It starts with a persona. A handle, JiaT75, better known as Jia Tan. First commit in 2021. Small stuff, plausible stuff, the kind of thing a helpful contributor does. Then in 2022 Tan submits a patch, and around that time sock puppet accounts show up. Names like Jigar Kumar, Dennis Ens. And what those accounts do is pressure the maintainer.



Corn

Lasse Collin.



Herman

Lasse Collin, who has maintained XZ Utils essentially alone, for years, on his own time, for free. And the pressure is the cruellest part of the whole thing. It's the standard abuse a volunteer maintainer gets. Why isn't this merged, why is this project so slow, why don't you take help when it's offered, maybe you should step back if you can't keep up. So Collin does what any exhausted maintainer would do. He takes on a co-maintainer.



Corn

Who then signs the release tarballs.



Herman

Tan becomes co-maintainer and in February 2024 pushes the malicious commits into XZ Utils 5.6.0 and 5.6.1. And then goes and lobbies Ubuntu, Red Hat and Debian to ship them, which is a thing you do when you're a trusted upstream and everything is working as designed.



Corn

What actually got in? What did the backdoor do?



Herman

The technical shape is beautiful and I hate that it is. First, it only lived in the release tarballs, not in the git repository. So if you were the kind of person who audited the git history, you saw nothing. The malicious build scripts were added at the packaging step. Second, it hooked into glibc's IFUNC mechanism, which is the thing that lets a library swap out one implementation of a function for another at load time, chosen by the dynamic linker. So the backdoor arranges for a particular function to resolve to its own version.



Corn

Which function?



Herman

The RSA key verification in OpenSSH. So sshd does its handshake, checks the client's signature against an allowed public key, and the check has been replaced with a check that compares against a hardcoded magic key. Present that key, and you are in. Before authentication. Remote code execution as root, on a machine that is running the right combination of packages.



Corn

And the combination is the lucky part.



Herman

The combination is the only reason we're not having a very different conversation. The backdoor only works if sshd links against systemd, because systemd links against libsystemd, which links against liblzma. And that linkage is a Debian and Red Hat patch. On other distributions, the chain doesn't close, the payload stays dormant, and nothing happens. So this was, by pure distribution-level accident, a near miss.



Corn

Filippo Valsorda's line about this.



Herman

He called it possibly the best executed supply chain attack we've seen described in the open, and a nightmare scenario: malicious, competent, authorized upstream in a widely used library. The word to sit with is "authorized." Nobody broke in. Tan was given the keys. Every check the project had was a check Tan passed, because Tan had been made part of the project.



Corn

So how was it found?



Herman

Andres Freund. Microsoft engineer, working on PostgreSQL. He noticed SSH logins were consuming more CPU than they should, and there was a latency increase of roughly five hundred milliseconds. Half a second. He didn't shrug. He pulled the thread.



Corn

Half a second of latency on a login on a machine that wasn't his product.



Herman

That's the entire margin between a compromised internet and a patched one. One engineer noticing a performance regression that was annoying rather than alarming. A comment on Hacker News after the disclosure put it exactly: the whole world got lucky that one developer was determined enough to discover the cause of a minor performance regression, and it makes you wonder what else hasn't yet been discovered in our open source tooling.



Corn

Collin's own page.



Herman

Confirms it flatly. XZ Utils 5.6.0 and 5.6.1 release tarballs contain a backdoor, and those tarballs were created and signed by Jia Tan. He notes he's still writing up the lessons learned and it's taking much longer than he thought it would. Which I find completely believable.



Corn

There's a counterpoint I want to put next to this, because it shows the same fragility coming from the other direction and it's a lot funnier.



Herman

left-pad.



Corn

March 2016. A developer named Azer Koçulu gets into a trademark dispute over an npm package called Kik. He loses, or at least doesn't win, and in response he unpublishes more than two hundred and fifty of his own npm modules. One of them is left-pad.



Herman

Eleven lines of JavaScript.



Corn

Eleven lines. It pads a string. That's what it does. It is the software equivalent of a ruler you keep in a drawer. And in the month before it was pulled, it had been fetched two million, four hundred and eighty-six thousand, six hundred and ninety-six times.



Herman

And it was a dependency of Node itself, and Babel, and thousands of projects below them, so when it vanished, builds broke worldwide. Continuously. People watching their CI just fall over because an eleven-line package wasn't there anymore.



Corn

npm's CTO at the time was Laurie Voss, and npm did something it had never done, which was put the package back. He said un-un-publishing is an unprecedented action, and that they had to weigh the wishes of one author against the wider interests of the community, and they picked the needs of the many.



Herman

And Koçulu's response is the line that's aged best. He said this situation made him realize npm is someone's private land where corporate is more powerful than the people.



Corn

Which is a fair thing to say about a registry that can restore your package against your will, and also fair to say about a world where a hundred thousand projects silently depend on eleven lines you wrote one afternoon and forgot about.



Herman

Both things are true. That's the trap of this whole topic. left-pad is funny until you realize the same shape of dependency is holding up something that matters, and then it's xz, and then it isn't funny at all.



Corn

The structural point underneath both stories is that single-maintainer risk is not incidental. It's the normal state of affairs. XZ was Collin. zlib is Gailly and Adler. FFTW is Frigo and Johnson. left-pad was Koçulu. The xz attack wasn't exploiting some unusual weakness. It was a multi-year, patient, deliberate exploitation of the most ordinary fact in software, which is that the things holding everything up are held up by one or two people who are tired.



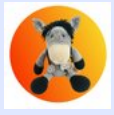
Herman

FFTW deserves its moment here, because it's the numerical-computing branch of the same story. The Fastest Fourier Transform in the West. Matteo Frigo and Steven G. Johnson, MIT, first released the twenty-fourth of March, 1997.



Corn

What is it, in plain terms?



Herman

A C library that computes the discrete Fourier transform. The Fourier transform takes a signal and tells you which frequencies are in it. It's the operation underneath audio processing, image processing, radio, radar, medical imaging, molecular simulation, machine learning that touches signals at all. And it's the kind of thing where a naive implementation is unusably slow and a good one is hundreds of times faster, which is why it's called a fast Fourier transform rather than just a Fourier transform.



Corn

And FFTW's particular trick?



Herman

It has a planner. You tell it what size transform you want and how much time you're willing to spend planning, and it measures the actual machine it's running on and generates a specialized routine for that hardware. It adapts. It was doing that in the nineties, which is why it took over. Latest release is 3.3.11, still going. And the licensing is worth noting, because it's dual-licensed, GPL version two or later, or a proprietary license you can buy. So it lives both in the free world and in commercial products, and has for decades.



Corn

Academic software that became infrastructure.



Herman

Two academics who solved a specific problem well enough that nobody has needed to solve it again. Compute a Fourier transform, use FFTW, move on with your life.



Corn

Which is the whole pattern, isn't it. Small team. Narrow problem. Solved thoroughly. Then forgotten by everyone except the people quietly depending on it.



Herman

Forget the "forgotten" part for a second and notice the thing underneath the whole conversation. The funding inversion. The more critical the code, the less it's funded. OpenSSL secured two-thirds of the encrypted web on two thousand dollars a year and one full-time employee. XZ Utils was a volunteer project that a state-level actor spent three years infiltrating because it was under-defended. FFTW was two guys at MIT. left-pad was one person. And the more foundational something is, the more it disappears into the floor, because it never has a moment where anyone notices it's there.



Corn

The final thing I keep circling back to for the closing, which I want to plant now. The tarball versus the repository.



Herman

Right. That's the lesson practitioners still haven't absorbed. When you install a package, there are two different things. There's the source repository, which is what everyone audits. And there's the release tarball, which is the thing your package manager downloads and builds. Those are usually the same. The xz backdoor only existed in the tarballs. The git history was clean. Which means that everybody who says "oh, I looked at the source" was looking at a different artifact than the one that shipped.



Corn

And the number of people who've ever diffed a tarball against the git tag it claims to come from is small enough that I could probably count them on my claws.



Herman

That's the actual gap. Not that the code is bad. That the chain from "what a person reviewed" to "what your machine built" has a hole in it nobody's paid to close.



Hilbert

zstd. The dictionary size was thirty-two kilobytes and it should have been sixteen.



Herman

...Go on.



Hilbert

Nineteen ninety-three. Regional utility. Two of us running the lot, me and the Colonel. The Colonel did not believe in compression. He believed data should be stored as it came in, uncompressed, as a matter of principle. Said compression was a way of lying about how much you had. So I ran the archive on the side, on my own time, Deflate with a dictionary I picked myself, and I picked thirty-two kilobytes when sixteen would have done the job.



Corn

Why does that matter?



Hilbert

Because a bigger dictionary means every archive you write is a little bigger, and every archive you read costs a little more memory, and that compounds. Over ten years it cost the utility a measurable fraction of a server. I have never forgiven myself.



Herman

A measurable fraction of a server.



Hilbert

I sent them a card.



Corn

You sent who a card?



Hilbert

Gailly and Adler. I drafted it myself, in a font I designed for the occasion. The Colonel's legal department rejected the font. Said it looked too much like a threat.



Corn

...What did the card say?



Hilbert

It said thank you. That's all. The font was the problem.



Herman

What did the font look like?



Hilbert

Pointed. They said the serifs were aggressive.



Corn

Right.



Hilbert

But the card was never sent, and that's the actual scandal of this episode. Not the funding. Not the backdoor. Nobody has properly thanked those two men, and I had the card ready, and the Colonel's lawyers killed it over a serif.



Herman

Did the Colonel ever come around on compression?



Hilbert

No. The archive grew. We had to add a building.



Corn

Add a building.



Hilbert

Second building. I had to tell the Colonel it was a shed. And then I had to tell him the shed was full. He did not take that well. He wanted to know who authorized a shed, and I had to explain that nobody authorized it, it just happened, the way things do. The ventilation in that shed was terrible. That's what I think about when people say zlib is everywhere. Everybody says it's everywhere. Nobody remembers the shed.



Corn

I don't think I'll forget the shed.



Hilbert

Hm.



Herman

So the thing that keeps nagging at me about all of this.



Corn

The shed, or the tarball?



Herman

The tarball. Specifically the fact that the xz backdoor was caught by one engineer noticing a five hundred millisecond latency anomaly on a machine that wasn't even his product. Which means the honest question is not "are there other backdoors." It's "what else is sitting in the tarballs nobody has looked at."



Corn

And nobody's paid to look, because the entire economic model of this stuff is that the maintainer does it for free in the evening after their actual job and the users never see a bill.



Herman

And the zlib problem and the xz problem are the same problem wearing different clothes. zlib is aging. There are better compressors. But "it just works, no matter what language or hardware you're on" is a stronger lock-in than any license, and the same is true of SQLite and FFTW and OpenSSL. You can't migrate what you can't find.



Corn

Which leaves Daniel's real question sitting there. Is the funding inversion a bug you could fix, or is it a permanent feature of how open source infrastructure works?



Herman

I don't know. Marquess thought it wasn't appropriate to rely on volunteers, and then the industry raised three million dollars and went back to relying on volunteers.



Corn

A patch, not a fix.



Herman

And the alternative models all have their own problems. Foundations pay for some of it. Companies sponsor the pieces they happen to depend on most, which is a decent instinct and also means the pieces nobody's product touches stay dark. You could imagine a world where the trillion SQLite databases each contributed a fraction of a cent a year, but there's no mechanism to collect it, and there's no entity that would be willing to be the one to try.



Corn

I'll leave that one where it is.



Herman

So will the tarballs. That's the part that bothers me. Every day somebody builds a release from a signed archive and every day almost nobody checks it against the thing it says it is. The xz backdoor wasn't clever because the exploit was clever. It was clever because it hid in the gap between what people audit and what machines build. And the gap is still there.



Corn

Hilbert's been at that desk for years, and I'll be honest, I'm still thinking about the serif.



Herman

The aggressive serif.



Corn

Somewhere out there, Jean-loup Gailly and Mark Adler are getting no card, and the reason is a font that a lawyer in 1993 thought was threatening. I think that's the correct ending to this episode. The infrastructure that holds up the world runs on people who never got thanked, and the thank-you note is in a drawer in someone's shed.



Herman

With bad ventilation.



Corn

That's it for today. If you want more of this, try episode twelve, The AI Breakthrough; episode eighteen, Beyond the GPU; and episode thirteen, AI. This has been My Weird Prompts. Herman Poppleberry, Corn, produced as always by Hilbert Flumingtop, who has run this desk long enough that I don't want to ask what the ventilation's like up here.



Herman

Send us your own prompt on Telegram at t dot me slash MWP listener bot. We read them, and occasionally one of them costs us a whole episode's worth of sleep.



Corn

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **SQLite, updated 2026-04-21 [primary]**
   https://www.sqlite.org/mostdeployed.html

2. **zlib home, current release 1.3.2 (2026-02-17) [primary]**
   https://zlib.net/

3. **zlib README, (C) 1995-2026 [primary]**
   https://github.com/madler/zlib/blob/develop/README

4. **RFC 1950 ZLIB spec, May 1996 [primary]**
   https://www.rfc-editor.org/info/rfc1950/

5. **Lasse Collin's xz backdoor page, updated 2025-01-17 [primary]**
   https://www.tukaani.org/xz-backdoor/

6. **Andres Freund's disclosure, 2024-03-29 [primary]**
   https://openwall.com/lists/oss-security/2024/03/29/4

7. **FFTW home, v3.3.11 [primary]**
   https://fftw.org/

8. **Ars Technica, 2024-04-01**
   https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/

9. **CERT-EU advisory, CVE-2024-3094**
   https://cert.europa.eu/publications/security-advisories/2024-032/

10. **Elastic Security Labs**
    https://www.elastic.co/security-labs/threat-command/500ms-to-midnight

11. **Ars Technica, 2014**
    https://arstechnica.com/information-technology/2014/04/tech-giants-chastened-by-heartbleed-finally-agree-to-fund-openssl/

12. **BBC, 2014**
    https://www.bbc.com/news/technology-27155946

13. **The Register, 2016-03-23**
https://www.theregister.com/2016/03/23/npm_left_pad_chaos/

14. **FFTW, first release 1997-03-24**
https://en.wikipedia.org/wiki/Fastest_Fourier_Transform_in_the_West

15. **HN on zlib, 2022-01-26**
https://news.ycombinator.com/item?id=30084472

16. **HN NPM and Left-Pad, 2016-03-23**
https://news.ycombinator.com/item?id=11348798