

# Security

**CS499/579 :: Empirical Computer Security**

**Zane Ma (he/him/his)**

**2024.10.07**

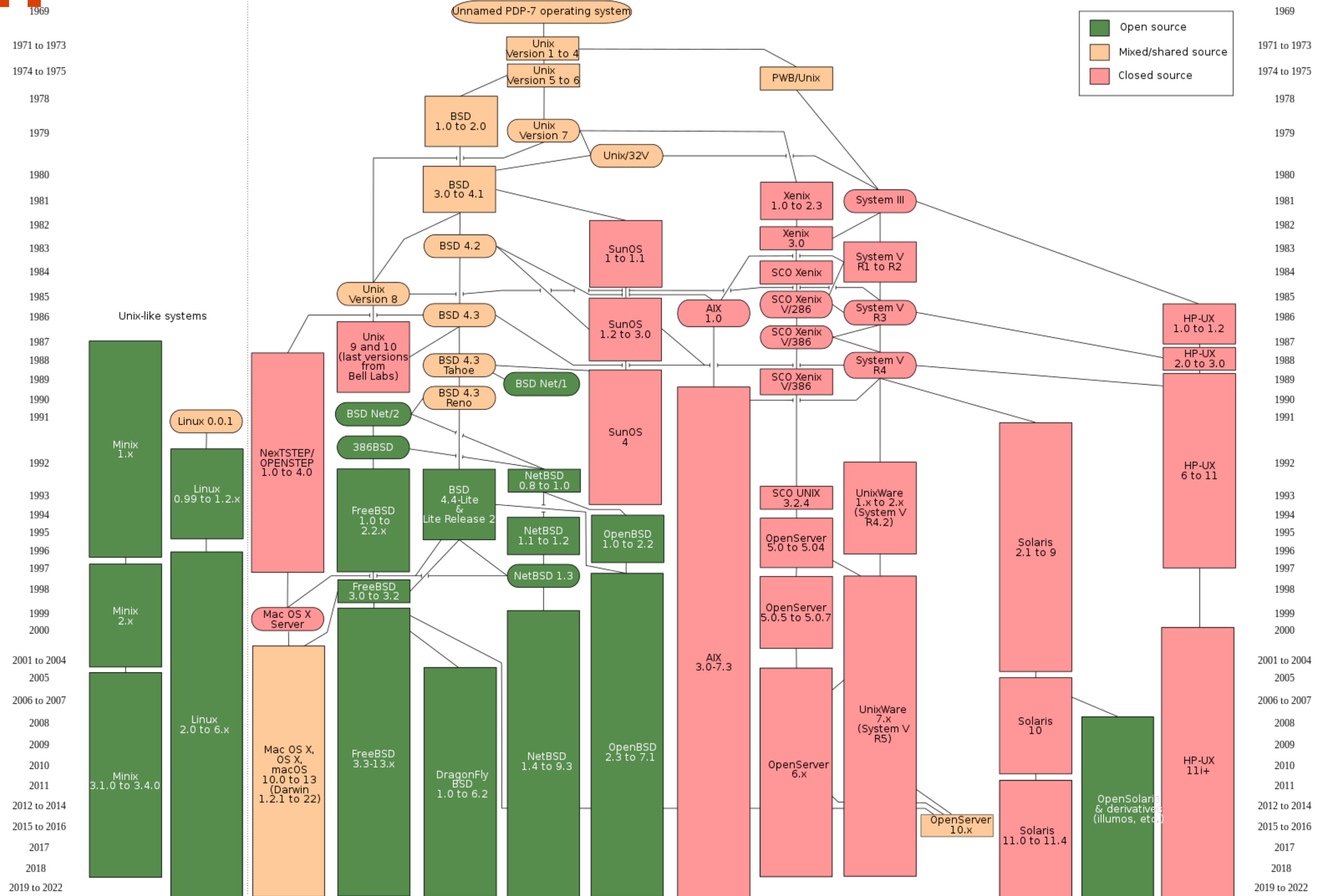
# Topics

- Trusting Trust - why is security hard?
- Case study: xz utils
- Authentication, Authorization, Auditing

# Ken Thompson



Co-creator of UNIX  
and Golang



# Reflections on Trusting Trust

```
char *s;  
{  
    if(match(s, "pattern")) {  
        compile("bug");  
        return;  
    }  
    ...  
}
```

How to detect?  
Look at compiler source code!

hello.c

```
void main() {  
    print("Hello");  
}
```

Human-readable code

compiles  
into

```
1010101000001111001  
0001011111000010111  
1011100001010000111
```

Compiler

hello.exe

```
1010101000001111001  
0001011111000010111  
1011100001010000111
```

Machine code

# Reflections on Trusting Trust

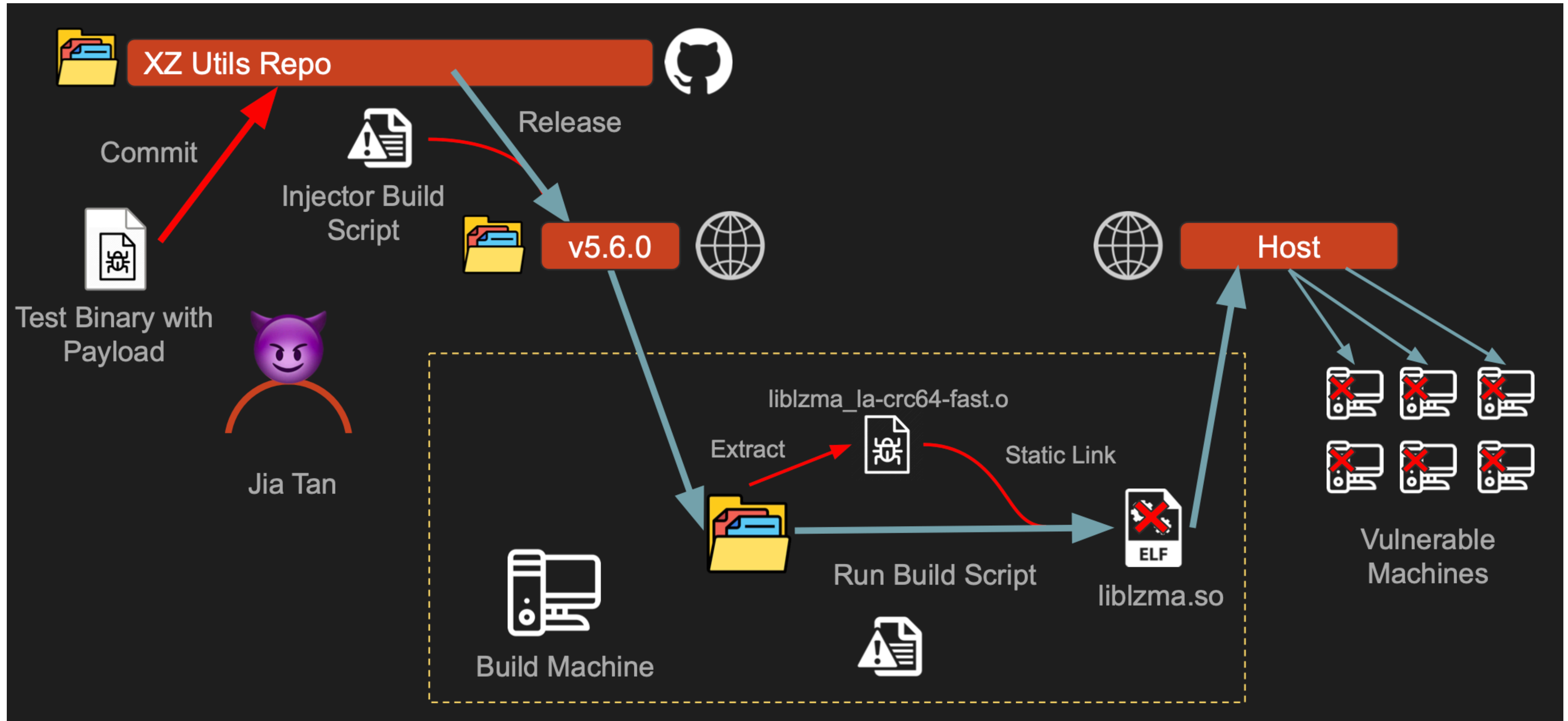
```
char *s;
{
    if(match(s, "pattern1")) {
        compile ("bug1");
        return;
    }
    if(match(s, "pattern 2")) {
        compile ("bug 2");
        return;
    }
    ...
}
```

- Pattern 1 = login operation that becomes insecure when compiled with bug
- Pattern 2 = compiler; anytime this compiler is compiling a future version of the compiler, it will inject the two matching patterns on the left
- Compiler binary contains both pattern 1 and pattern 2 bugs, in perpetuity, even if we remove them from the compiler source code!
- tl;dr - self-perpetuating vulnerability-injecting compiler that only exists in the machine code binary and cannot be seen from source

# xz utils backdoor

- xz = open-source suite of compression software included in nearly all Linux/Unix-like systems; used by popular software, including OpenSSH
- February 2024: malicious code released, but not obvious in main open source code! Inspecting git repository does not reveal maliciousness
  - Malicious payload hidden in benign-looking test binary (e.g., random data file to test compression algorithm on)
  - Non-reproducible build: Released software (tar) not fully derived from git repository; stealthy addition of injector script
  - Malicious code is not linked until build time, on third-party machines

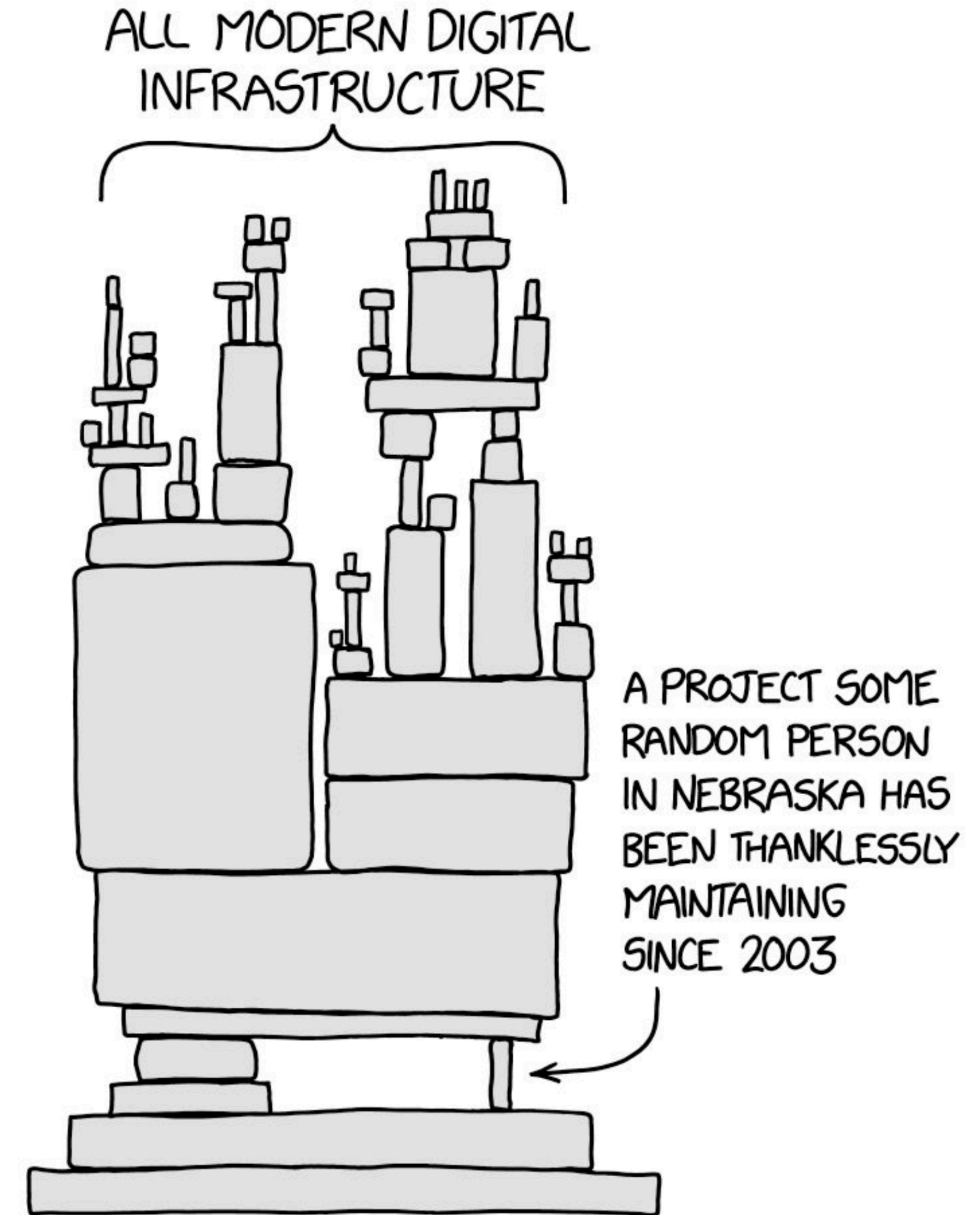
# xz utils backdoor



<https://cs4157.github.io/www/2024-1/lect/21-xz-utils.pdf>

# xz utils backdoor

- A determined, capable adversary
  - 2.5-year-long OS contribution history
  - “Multi-actor” social engineering campaign
  - Technically advanced stealthy attack
- Supply-chain attack would have provided remote root access to millions of internet hosts, fortunately discovered March 2024 before deployment in mainstream systems

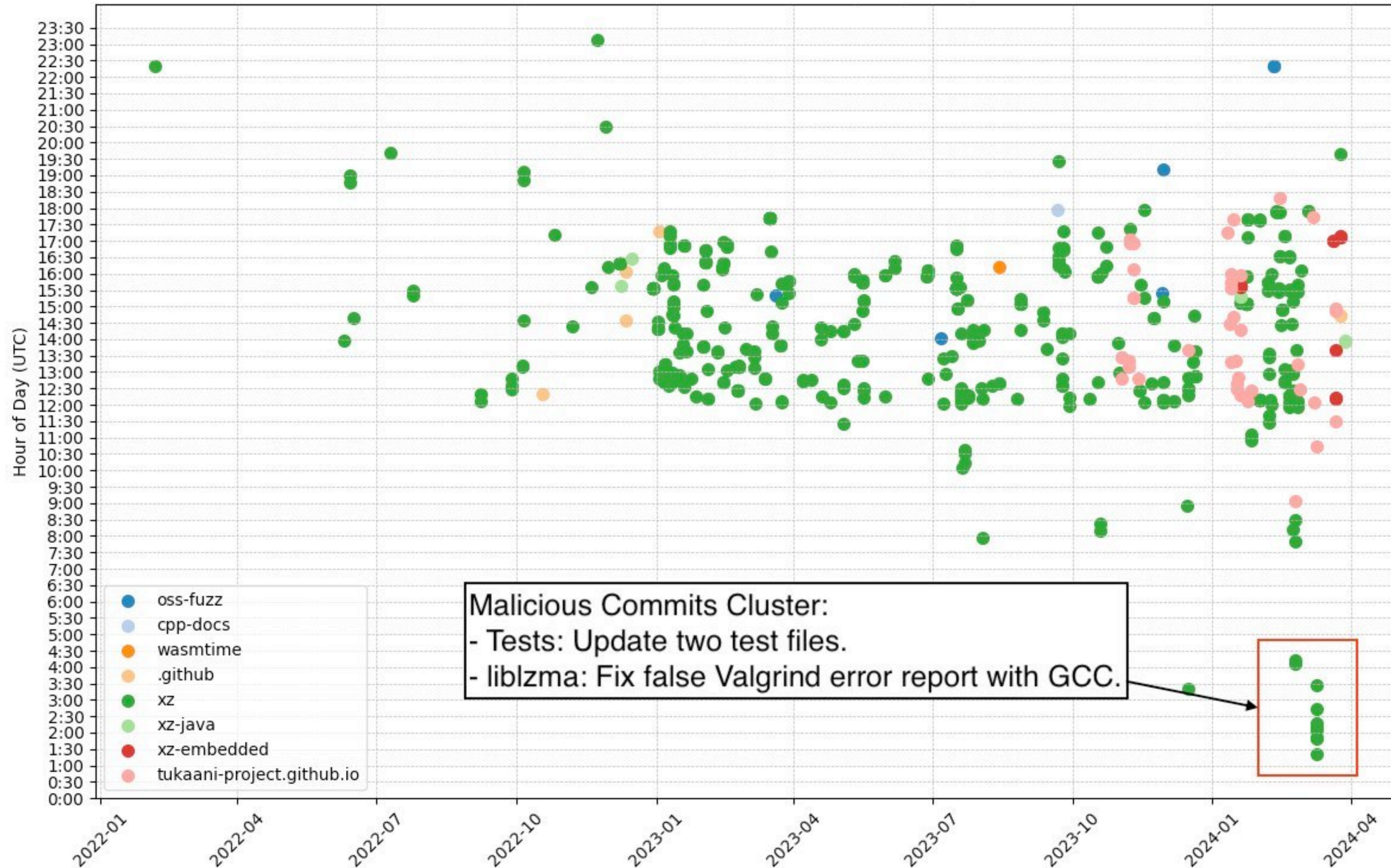




# Who is Jia Tan?

- Active open-source contributor, began contributing to xz in 2021
- Hundreds of commits to the library over the course of 2-3 years
- Ultimately promoted to co-maintainer of xz utils in 2024
  
- Once privileged, they added two “test” files to the xz repo:
  - bad-3-corrupt\_lzma2.xz
  - good-large\_compressed.lzma
- Also added malicious payload injector line to released build scripts

# Who is Jia Tan?

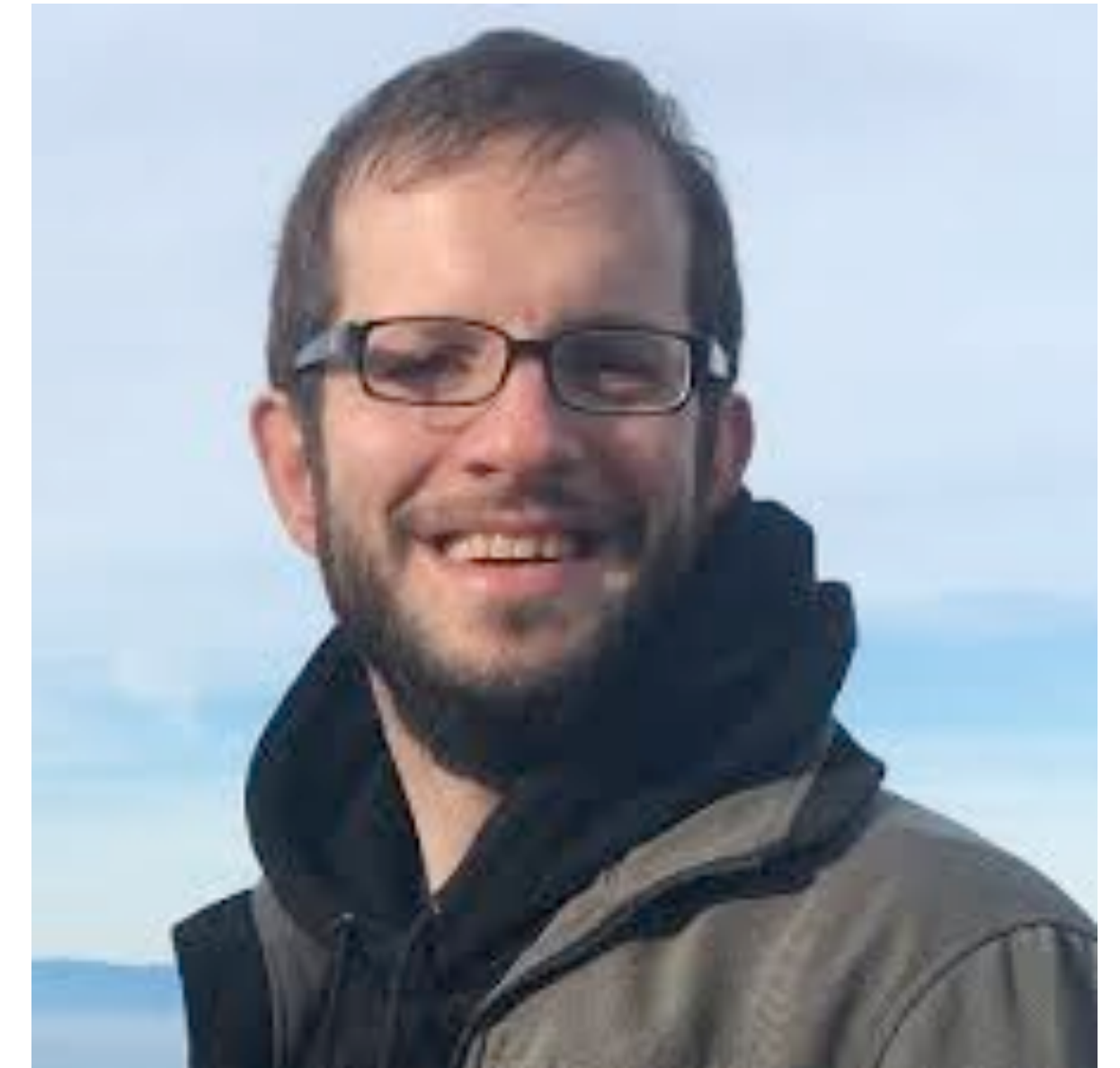


# xz utils - Social Engineering

- Maintained by Lasse Collin for many years - solitary and thankless job
  - “Your efforts are good but based on the slow release schedule it will unfortunately be years until the community actually gets this quality of life feature.”
  - “Progress will not happen until there is new maintainer[...] The current maintainer lost interest or doesn't care to maintain anymore. It is sad to see for a repo like this.”
  - Many other comments from three accounts (Jigar Kumar, Dennis Ens, Hans Jansen) to 1) pressure Lasse hand over control / make Jia Tian an xz maintainer, 2) update OS versions of xz-utils with the newest/malicious version

# xz utils - lucky discovery

- Andres Freund - Microsoft PostgreSQL dev
  - Found odd performance behavior on Debian test release
  - Noticed failed logins taking a long time...should be fast!
  - “With the backdoored liblzma installed, logins via ssh become a lot slower. [...] before: real 0m0.299s [...] after: real 0m0.807s”
  - Malicious code released Feb 2024 —> discovered Mar 2024; if no latency differences, it could be deployed and undiscovered even today!



# xz utils - Takeaways?

- Specific technical exploit considerations: reduce dynamic linking dependencies (i.e., sshd shouldn't link liblzma), reproducible builds - ensure released code (tarballs) match repository source
- Supply chain attacks are challenging problem; problematic for critical infrastructure to depend on anonymous developers
  - Critical infrastructure relies on the same (large) set of open-source software
  - Software Bill of Materials (SBOM) - list of all software dependencies; good starting point but difficult to ensure accuracy, mostly a forensic capability
- xz utils is not the first, nor last supply chain attack - next one will avoid noticeable latency issues

# Reflections on Trusting Trust

```
char *s;
{
    if(match(s, "pattern1")) {
        compile ("bug1");
        return;
    }
    if(match(s, "pattern 2")) {
        compile ("bug 2");
        return;
    }
    ...
}
```

- Pattern 1 = login operation that becomes insecure when compiled with bug
- Pattern 2 = compiler; anytime this compiler is compiling a future version of the compiler, it will inject the two matching patterns on the left
- Compiler binary contains both pattern 1 and pattern 2 bugs, in perpetuity, even if we remove them from the compiler source code!
- tl;dr - self-perpetuating vulnerability-injecting compiler that only exists in the machine code binary and cannot be seen from source

# Reflections on Trusting Trust

## **MORAL**

The moral is obvious. You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me.) No amount of source-level verification or scrutiny will protect you

- Can't trust compiler —> verify correctness of compiler beforehand or write your own from machine code
- Can't trust processor to execute code properly —> test hardware / drivers beforehand or create your own processor, and so on...
- Can't trust anyone implies... ***do everything yourself, from scratch.***

# BLOOKET BREAK



# Less daunting alternative?

- Accept the impossibility of perfect, guaranteed security - rely on trust!
- This is how modern society works
  - Trust government regulation - food from the store is safe to eat
  - Trust societal norms / laws - drivers won't act erratically
  - Trust friends, family - help you do things

Trust is imperfect - no guarantees, but it's more realistic than the alternative.

# Trust enforcement

- Trust X to do A. If they don't, you can:
  1. Choose not to trust X in the future (e.g., don't purchase from brand X, which produces low quality items)
  2. And/or punish X (e.g., going to jail for breaking the law)
- Trust on the internet is difficult because:
  - Inadequate authentication - can't determine who to trust / distrust
  - Insufficient regulation / laws - few repercussions for trust-breakers
  - Implicit trust - cannot automatically act on violations of trust

# Inadequate authentication

- Scenario: someone hacks your web server / phishes you / installs malware
  - IP-based network logs are insufficient to track down who did it
    - Tor network, VPNs, bulletproof hosting clouds don't track humans
  - Can't avoid the same actor next time, too easy to spin up an infinite number of new "network identity" - website names, IPs, phishing sites
- Scenario: you get scammed and your Bitcoin wallet is drained.
  - Even though all bitcoin transactions are "authenticated" with a cryptographic key pair, any one can spin up any number of Bitcoin identities and mix / wash the stolen funds

# Insufficient regulation

- For example, no laws against online abuse / harassment
  - The internet makes crime/abuse scalable, different enforcement considerations
- Even when there are laws, they are often just a “slap on the wrist”



- Thousands of customer’s raw DNA data exposed to the public —> \$75K fine
- The European Union has been leading the way: e.g., General Data Protection Regulation

# Implicit trust

- Even when authentication and legal consequences exist, we don't explicitly track who we are trusting for what!
- Supply-chain security
  - Both hardware and software supply chain
- **Provenance**: a record of ownership, used as a guide to authenticity or quality
  - System provenance: trace which processes communicate with each other, and what resources they access
  - Network provenance: trace which network hosts communicate, and what data they transmit to each other



# What about privacy?

- Crucial aspect; should be decided by society + legislation, not companies
- Privacy vs authentication is a nuanced spectrum
  - Potential starting point - digital equivalent of non-digital societal norms?
- Privacy vs accountability tradeoff: e.g., cash usage in the US; Tor darkweb
  - Challenge: privacy benefits individuals, privacy abuse can harm many

*Research project: characterize + quantify this tradeoff*

# AAA: Authentication, Authorization, Auditing

- Butler Lampson (1992 Turing Award winner)
- Premise = some system with sensitive / valuable resources; for example, website with user health info, power generator, memory of a VM / process
- Authentication: who is trying to access the resource
- Authorization: what the authenticated entity is allowed to do (read, modify)
- Auditing: a log of “Who did what when?” - for retroactive detection / forensics

# Why don't we have “real” security?

- Systems are complicated, so they have bugs
- People don't buy it
  - Danger is small, so it's OK to buy features instead
  - Security is expensive
    - Configuring security is a lot of work
    - Secure systems do less because they're older
  - Security is a pain
    - It stops you from doing things
    - Users have to authenticate themselves
- Goals are unrealistic, ignoring technical feasibility and user behavior

Butler Lampson. “Perspectives on Security.” *SOSP*, 2015.



# TODOs for you

First paper reading + questions will be due by 6PM **Tuesday, October 8th.**

Get the creative juices flowing! Project proposals (1-page max, at least 10pt font, single spaced) due **9PM Wednesday, October 16th.**

If you feel stuck, please come discuss ideas at office hours, or schedule time to chat with me

I will read all the proposals and meet with all teams for 30 minutes on *Monday, October 21* - please sign up for a meeting! Link on website