Lecture 35 — DevOps, but Operations

Patrick Lam & Jeff Zarnett
patrick.lam@uwaterloo.ca jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

July 14, 2024

Devops, but Operations

Services are doing The Thing – but how do we know if there's a problem?

Monitoring and Alerting.

"Action stations. Set Condition 1 throughout the ship!"

What to Monitor?

What to monitor and what to do about it?

Applications have health checks, but how basic should they be?

Maybe automate testing the basic workflow.

These don’t show performance problems.

What To Monitor, For Real?

- CPU Load
- Memory Utilization
- Disk Space
- Disk I/O
- Network Traffic
- Clock Skew
- Queue lengths
- Application Response Times

Overview and Summary

It would be good to have a dashboard of some sort.

It’s a little late to recommend you take a course in the future...

Dashboards are cool but shouldn’t be used to identify alerts.

Alert Conditions

- CPU usage exceeding threshold for a certain period of time
- Increased rate of error logs over a period of time
- A service has restarted many times recently
- Queue length very long
- Taking too long to complete a workflow


I DON'T KNOW
WHAT WE'RE YELLING ABOUT

Sir Arthur Conan Doyle Nerdery

Remember also the lesson from Sherlock Holmes in “The Adventure of Silver Blaze” – the dog that did not bark was a clue as to who did the crime.

Customer Support

The final option for detecting a problem is customer support.

Automated monitoring maybe can’t find everything every time, but we shouldn’t be relying on this as the primary mechanism.

Red Alert

- **Alerts**: a human must take action now;
- **Tickets**: a human must take action soon
  (hours or days);
- **Logging**: no need to look at this
  except for forensic/diagnostic purposes.

Common bad situation: logs-as-tickets.

Alerting

We usually talk about it in the metaphor of pagers, as in the small rectangular box that wakes up doctors when patients need them.

The alert may contain some information about the event that has caused it.

Example: Service *A* CPU usage is high.

Beep Beep

What do you do when you hear the fire alarm?

Back to Sleep

We have experienced too many false alarms.

If there is an actual fire, you will not only be wrong, you might also be dead.

Not the Kittens!!!

Alerts and tickets are a great way to make user pain into developer pain.

Some SUPER CRITICAL ticket OMG KITTENS ARE ENDANGERED is an excellent way to learn the lesson...

Devs will take steps that keep these things from happening in the future.

Incident Reports

Learning from incidents requires reports (post-mortems).

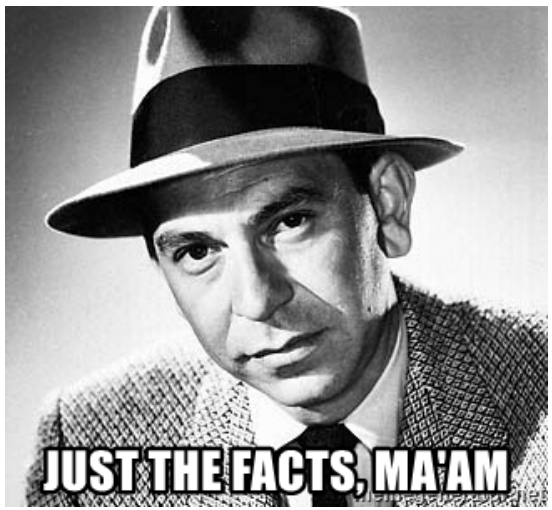
Identify root causes and what we can learn.

Reports are worth doing even if they aren’t fun.

Report Content

Breakdown of what happened with timeline.

What happened, when it was noticed, actions taken, when it was resolved.



Root Cause Analysis

There are two kinds of causes: root causes and proximate causes.

Example: deadlock.

How far do we go in terms of finding the root cause?

Toyota Methodology: Five Whys

Toyota model: ask “why?” five times to get to the root cause.

Does not have to be exactly 5; use judgement.

Too few times $\rightarrow$ superficial answers.

Too many times $\rightarrow$ “computers were a mistake.”

Action Items

What will we do in the short term and long term?

Prevent this from happening again.

Be Realistic

Action items have to be realistic...

Probably cannot say rewrite all code in Rust.

Not every report results in major head-exploding conclusions.

Not Included In Report

Avoid having:

- Irrelevant detail
- Speculation
- Blaming
- Blaming

Security, Report to the Bridge

Having an always-available service accessible over the internet makes security a very big concern.

Risk Enormous

You can run some program with tons of security vulnerabilities offline and feel that the security problems can be managed.

When it’s online the risk is enormous.

All kinds of vulnerabilities are a problem, but I’ll call out two of them:

Code execution/injection and data leakage (information exposure).

Abusing Free Services for Fun and Profit

Real-life attack about abusing free CPU time from cloud providers.

These are there for non-nefarious purposes but can be abused.

Also consider a scenario where attackers use your resources!

Abusing Free Services for Fun and Profit

Bypass signup limitations, browser scripting.

Mining cryptocurrency – inefficiently – but works.

Using $103 000 of resources produces one coin worth about $137: 0.13% return.
Or free money from attacker POV.

Abusing Free Services for Fun and Profit

Disguise what is happening: no container called `mine-crypto-lol` `linux88884474` instead.

Generate random Github action names.

Spotting crypto mining is easier because it's CPU intensive...
What about spyware, sending secrets to extortion rings?

Information Exposure

Information Exposure is bad for your reputation and expensive.

GDPR infractions get expensive: `https://enforcementtracker.com/`

Fines are for things like insufficient technical measures.

Fun Example

Remember talking about logging in tracing? Logging PII there was a big risk!

An example of a security process backfiring.

Audit This

There are some companies out there that will check your code for libraries with versions having known security vulnerabilities.

Sadly, when there is an updated version of a library, there may be breaking changes in it...

Checking for vulnerabilities should be an automatic process as part of your build and release procedures.

Supply Chain

Early in 2024, a vulnerability was discovered in the `xz` library.

The library itself is used for data compression but some versions of `openssh` rely on it.

This is a strong example of what's termed a "supply chain attack".

Supply Chain

Almost any modern piece of software is built in a compositional way.

Compromising dependency *F* means creating a vulnerability in *A*...

If *A* is `openssh` (*secure* shell daemon) $\rightarrow$ BIG problem.

How do you find it?

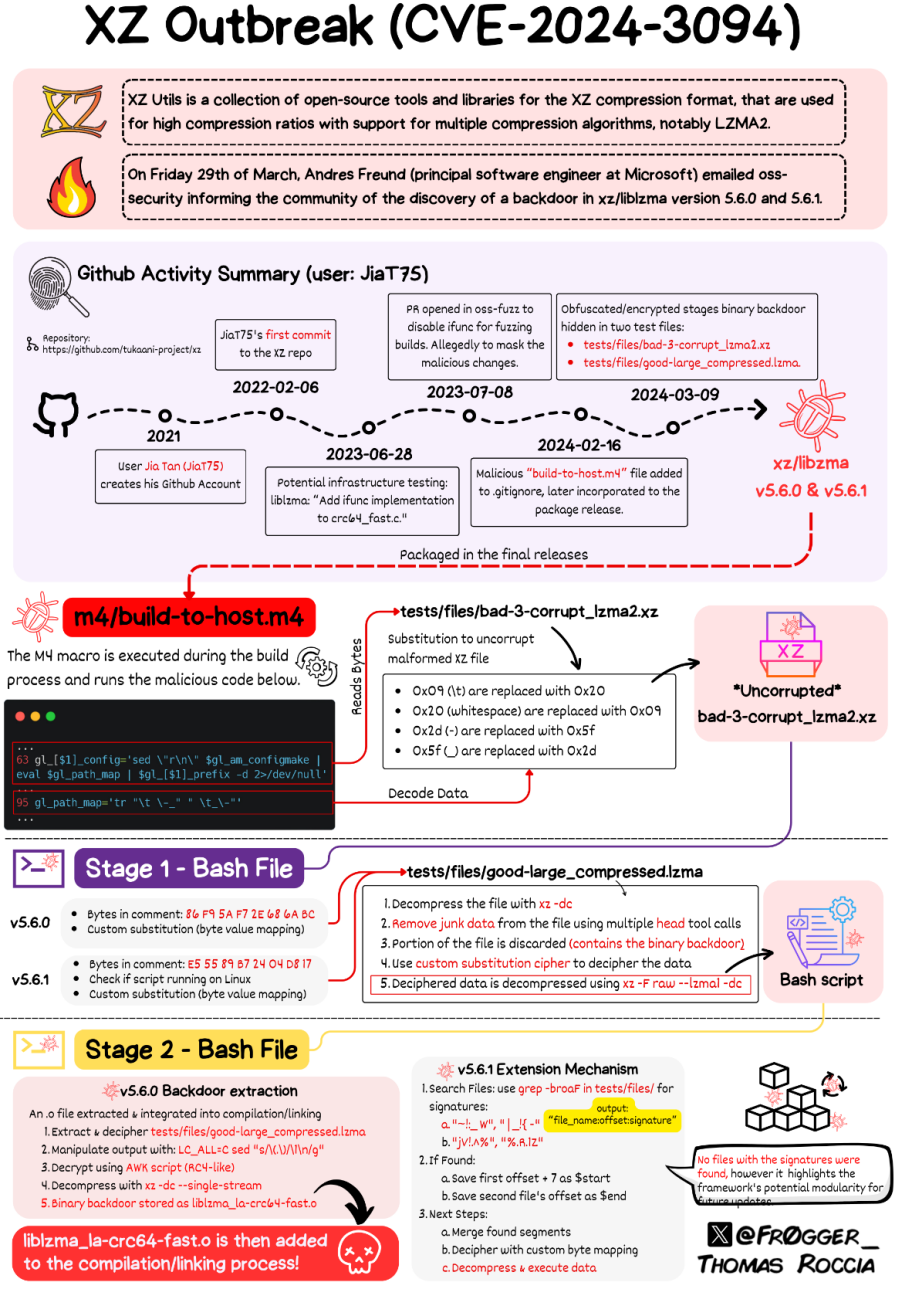
```
before:
nonexistant@...alhost: Permission denied (publickey).

before:
real 0m0.299s
user 0m0.202s
sys 0m0.006s

after:
nonexistant@...alhost: Permission denied (publickey).

real 0m0.807s
user 0m0.202s
sys 0m0.006s
```

The Vulnerability in xz Explained



How do you defend against it?

There’s obfuscation in the source code, so analysis is hard.

This also had a social engineering component.

How do you validate transitive dependencies?