Assuming the worst

Alexander Klimetschek

@aklimets

AEM Assets Operations WG

Do you want more sleep?

Do you want less angry customer interactions?

Yes.

We all want LESS critical escalations.

But they are not doing it right, it's their fault!

Maybe.

But that does not matter.

We have no choice.

Couple of reasons:

a) We need to support customers who are no experts.

Because many customers will be new to everything AEM.

Admins do not (yet) know AEM or Oak.

b) Developers and admins are not senior.

It's probably 80/20.

Candidates not hired by us or our partners will end up in our customers' IT department.

c) There is always an escalation cost.

First: figuring out what went wrong. Drains resources even if the customer is to "blame" in the end.

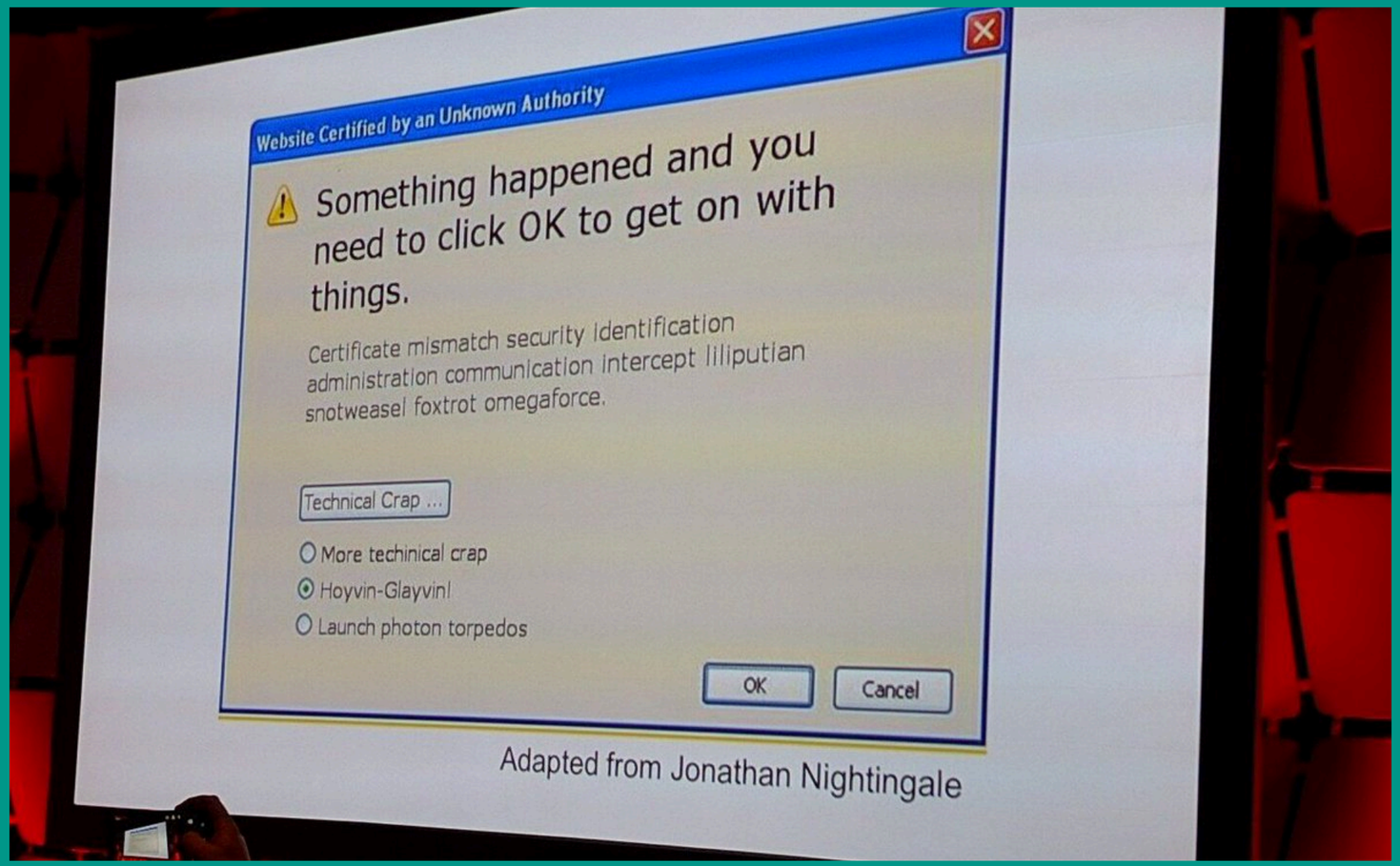
Second: getting them back on track. Can be costly if in production and for example the wrong Mongo version was installed.

d) To err is human.

Even our own Adobe folks can't be perfect all the time. (tbc)

Especially if you don't know everything about the system.

What users see in warning boxes:



Initial conclusion:

We should assume the "worst", not the "perfect".

Well, what is the "worst"? Examples?

Exhibit A:

Call save() after every property written. In a loop.

Exhibit B:

Reindex everything when a single index would have been enough.

Exhibit C:

Activate large asset folder, then deactivate subfolder right away to "exclude" it.

Exhibit D:

Concurrent asset workflows.

Exhibit E:

Upgrade with S3DS configuration even though it is a File data store.

Why did they do all these things?
Why was it possible to shoot oneself in the foot?

Exhibit F:

AWS S3 outage was a typo on the command line.

AWS answer: tools were not safe enough, fix them.

Exhibit G:

AGS recommended 2 Mongo nodes to customer even though doc says 3 is the minimum.

Yes, even Adobe AEM experts do not read the documentation :/

What does that mean for us?

First, we should think about what can go wrong.

Secondly, we should see how to prevent it.

Even if they do not RTFM.

It’s the difference between designing for the ideal case of how people *should* behave...

...versus designing for the practical case of how people *actually* behave.

Example:

If I run against an unsupported version of Mongo, will startup fail or is there a healthcheck warning?

Example:

Can I delete all checkpoints with no warning?

Example:

Will the system tell me before I start a reindex that it might take a week?

Example:

If I call save() too often, will the system be safe?

Example:

If I make a typo in the sidegrade command, and notice only later, can I recover?

Example:

What are best practices for adding custom assets processing?

Ok, I thought about it, and now?

There are different things we can do, with different amount of effort.

A. Documentation of best practices

B. Documentation of recovery options

C. Active monitoring and healthchecks

D. Chaos monkey tests, including horrible code

E. Check for critical inputs in admin tools

F. Incorporate recovery paths

G. Protect from misbehaving code

H. Have APIs that guide, and only allow good patterns

...Recap...

Three easy steps:

1. Assume a non-perfect user.

2. Imagine what can go wrong.

3. Document, design and implement protection.

With that I hope to have changed the way you look at customers, even if just a little bit.

Sleep well!

... one more thing ...

Game time!

1. I will present three short customer stories.

2. You pick one story and think of any way to prevent the problem.

3. You put that idea into the Connect Q&A chat pod. Include the story #.

For example, you might write:

"Story 1: The fabulator should prevent startup of the system with a big bang if connected to albimaster version < 3.0, so that customers can't even go into production with this setup."

Story 1

Customer support ticket:

Since our Linux box is out of space, and CQ was warning about it, I deleted some files under `crx-repository/repository/segmentstore`. After restarting the adobe, I see only the root directory in CRX. Please help!

Story 2

Issuing a query that seemingly returns nothing ended up deleting the entire `/apps` tree. Surprising, crazy customer code probably!

However: they were running a specific query that returned an asset path to be deleted. The query itself was perfectly fine - it only searched underneath an asset folder that was given as input, so there was no way this could point to /apps. However, under certain conditions that path extracted from the result was an empty string. And when you pass an empty string to `getResource(path)` you will get what? The Resource for the node at `/apps` ... because of the resource type resolution against the search path.

Story 3

Customer has an issue that any non-existing page URI will create directories on the dispatcher, being a great DoS attack vector on the dispatcher disk.

For example, requesting `http://example.com/anything/random` would create `/anything/random` with the not found error page in the dispatcher cache.

Turns out they have a custom 404 error page, which accidentally set the status code to 200 on the result due to some common utility code, hence tricking the dispatcher into caching the result.

Stories Recap

1. Deleted entire segmentstore directory to free up space on a full disk.
2. Query deleting all code because `getResource()` returned `/apps` for empty string.
3. 404 error page accidentally setting 200 http status, filling dispatcher cache.

Happy brainstorming!

Thank you for your participation!

Suggest you do similar exercises in your teams every now and then, sharing stories from escalations and brainstorm what can be done to prevent them in the first place.

If you are interested in more or have your own stories to share, visit:

https://wiki.corp.adobe.com/display/~aklimets/Fun+AEM+Customer+Stories

Thanks - over and out.