SYSADMIN

The Living Dead

TIM BRADSHAW

Tim Bradshaw was going to be a physicist before getting side-tracked by computers, in the form of UNIX and Lisp. He's worked on the border between programming and system administration for about 25 years. He worked in a large British retail bank during and after the crisis of 2007–2008 and is hoping not to be anywhere near one next time around. tfb@tfeb.org

It has been claimed [1] that system administration is dead: that may be so, but if you think that the dead are not a problem, you have been watching the wrong kind of movies. System administration may be dead, but it still walks.

The kind of system administration we'd like to be dead is *manual* system administration: the management of computing systems by people, rather than by programs. Not only is this very expensive but people are not up to the task: they make mistakes and forget things, so inconsistencies accumulate over time, leading to a slow collapse of the system. Long before the final collapse, it becomes impossible to deal with performance and security problems. These problems slowly kill organizations that try to run large systems using traditional approaches.

Why should we care? If we know how to do better now, then why not let organizations that cannot or will not learn fail? That's a solution only if we don't care about those organizations collapsing, and if we really do know how to do better.

I Walked with a Zombie

Consider an example of an organization that has solved the problem and one that hasn't: I'll pick Google for the former and a large retail bank of your choice for the latter. How bad would a week-long outage to each of these organizations be?

Google. This would be reasonably bad: search itself is a solved problem now—there are other adequate providers—but a lot of people have built their lives and businesses around products like Gmail without much thought. They'd have a bad week, and some businesses would die.

The Bank. This would be rather worse. If it was your bank, you would have no access to money at all other than cash you were carrying, and you would be hungry when that ran out. This may not be the end: banks are real-time organizations and can only be down for so long before they cannot recover, whether or not they have lost data. Opinions vary on how long this is but it's around a couple of days; your bank might never come back, and you would have to pick up the pieces of your financial affairs over months. This too may not be the end: the banking system is heavily interlinked, and the failure of a large retail bank could easily cause a cascade failure of other banks. The correct defense against that involves canned food, firearms, and growing a beard: you *do not* want a banking collapse to happen.

A failure like this is not easy to fix: you can bail out a bank that has run out of money by pouring money into it, but you can't bail out a bank whose computer systems have failed by pouring computers into it. We also should not assume that "someone else" will magically fix it for us. If the people who regulate banks were not competent [2] to see a rather obvious financial crisis coming in 2007–2008 until it was too late, they certainly are not competent to spot crises in computer systems, let alone fix them. And crises do happen. The 2012 RBS batch problem [3] was a damned near-run thing, and there is every reason to believe that something like it will happen again in a bank or some other equally critical organization.

Perhaps a banking collapse is not very likely, but it is definitely *possible*, and even a fairly small risk of the apocalypse is a thing to be avoided. Poor system administration practices matter, and it is not enough to declare them dead. We actually have to do something to stop them lurching around eating us.

Bone Sickness

We understand and can solve many of the roots of this malaise. Structural and funding problems and organizations outsourcing their own brains result from incompetent management. Administrators who do not program and programmers ("developers") who do not administrate lead to the problems you would expect, the solution to which is some variant of DevOps. Additionally, the people who build and deliver systems should be *the same people* who later maintain them; throwing rubbish over the wall is less appealing if you know you will have to clean it up later.

But these are not the only difficulties.

Old. The organizations I'm worrying about are *old*, which means they are not growing exponentially. All exponentially growing organizations are, effectively, young (although not all young organizations grow exponentially). Exponential growth famously kills companies, but there's a converse: if you *can* handle it, then all other problems are easy because all your mistakes get inflated away. Organizations growing exponentially can simply ignore old systems. Unfortunately, and despite what economists pretend to believe, exponential growth is necessarily ephemeral.

Contracts. If you have contracts with teeth, then you can't just turn off the system that is supporting a contract when it suits you. Dealing with this requires either applications and languages that are compatible for many years or physically supporting very old machines. The "Cascade of Attention-Deficit Teenagers" development model [4] means that the second option is often less bad, and we should be ashamed of this. Exponential growth inflates this problem away as well, while it goes on, but avoiding meaningful contracts is a clever trick if you can do it. Banks, sadly, are entirely made of contracts.

It is interesting that the canonical "good" organizations are exponentially growing, have avoided contracts with any real bite, and indeed do simply turn off services [5] when it suits them. Whether they really have solved the system administration problem will become more clear as their growth slows and contracts start to bite in the coming few years.

Power and Safety. To solve the system administration problem, you need powerful tools: tools that can influence very many machines, and tools that may themselves be computationally powerful and, hence, have behavior that is hard to reason about. Name services are an example of the former, and systems that can run arbitrary code on many machines, such as configuration management or patch deployment systems, are examples of the latter.

Such tools have inherent safety problems.

To start with, you need to be sure that whatever you are doing is either correct, does no harm if it is not correct, or, if harm is done, is fully reversible. Related to this are questions about authority and auditability: if you work for the sort of organizations we're discussing, you need to be able to show that you have authority to do something and later demonstrate convincingly to auditors that you had authority, that you actually did the thing you had authority to do, and so on.

Both of these problems exist already: a very powerful system simply makes them enormously more serious. It's the difference between the precautions you would take handling a stick of dynamite and handling an MK-53. These problems are mostly solvable in principle, although I don't think they are very close to being solved in practice. One non-solution is to divide the system up by some security mechanism so that large changes can't be made; well, yes, but then you will need lots of administrators for all the divided chunks, which is where we came in.

There is a graph that describes control and authority in a system: root nodes and nodes near them are extremely sensitive, as a compromise of them is a compromise of the system. Understanding the graph and working a lot harder on the security of the programs and protocols that sit at or near the roots of it would be a good start at dealing with this. Unfortunately, understanding the graph tells you one enormous thing: the roots are people and buildings, all of which can be attacked in very traditional ways, and the 2014 Sony attack [6] seems to be an example of that.

There are parts of the graph beyond any given organization that can themselves be compromised. For instance, the kernel.org compromise [7] was only not serious because it was discovered quickly and because of good engineering practices. Generally, there is blind faith that "vendor code," while probably buggy, won't be intentionally compromised or, if it is, only by the good guys [8]: why do we think that, since that code is right at the roots of the graph?

Banks are forced to care about these questions, and they are encrusted with auditors whose job is to make them care. They only have answers to some of them, and their answers tend to involve a deeply hideous bureaucracy. That very bureaucracy makes it extremely hard for them to think clearly about underlying problems such as the control and authority graph. This is particularly alarming given the sorts of configuration-management tools that they are being sold.

Warm Bodies

It is not just banks that are vulnerable, but utility companies, air traffic control, and governments: every organization whose failure would be most damaging. And bludgeoning the zombies isn't enough, since the problem is not really solved at all, other than in some rather special and almost certainly ephemeral cases. Do we really know how to manage large systems *in general* in a way that is demonstrably safe? Do we know how to build large systems that are safe at all? I don't think we do, not least because *really general solutions do not exist*. And claiming that we have solved problems that we, in fact, have not solved at all will simply suppress efforts to find answers that might be good enough. The obituary of system administration has been written prematurely.

References

[1] Todd Underwood, "The Death of System Administration," *;login:,* April 2014: https://www.usenix.org/publications/login/apr14/underwood.

[2] Sewell Chan, "Financial Crisis Was Avoidable, Inquiry Finds," *The New York Times*, January 25, 2011: http://www.nytimes.com/2011/01/26/business/economy/26inquiry.html.

[3] John Campbell, "Ulster Bank IT Problems: What Went Wrong," BBC News, November 20, 2014: http://www.bbc.co.uk/news/uk-northern-ireland-30127164.

[4] Jamie Zawinski, "The CADT Model," 2003: http://www.jwz.org/doc/cadt.html.

[5] "A Second Spring of Cleaning," Google, March 13, 2013: http://googleblog.blogspot.com/2013/03/a-second-spring-of-cleaning.html.

[6] Bruce Schneier, "Comments on the Sony Hack," December 11, 2014: https://www.schneier.com/blog/archives/2014/12/comments_on_the.html.

[7] Jonathan Corbet, "The Cracking of kernel.org," August 31, 2011: http://www.linuxfoundation.org/news-media/blogs/browse/2011/08/cracking-kernelorg.

[8] Bruce Schneier, "More About the NSA's Tailored Access Operations Unit," December 31, 2013: https://www.schneier.com/blog/archives/2013/12/more_about_the.html.

LISA15

More craft.
Less cruft.

Nov. 8 – 13, 2015 | Washington, D.C.
usenix.org/lisa15

The LISA conference is where IT operations professionals, site-reliability engineers, system administrators, architects, software engineers, and researchers come together, discuss, and gain real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world.

Submit your ideas for talks, tutorials, panels, paper presentations, and workshops by April 17, 2015.

Topics of particular interest at this year's conference include systems and network engineering, monitoring and metrics, SRE/software engineering, and culture.

Sponsored by USENIX in cooperation with LOPSA