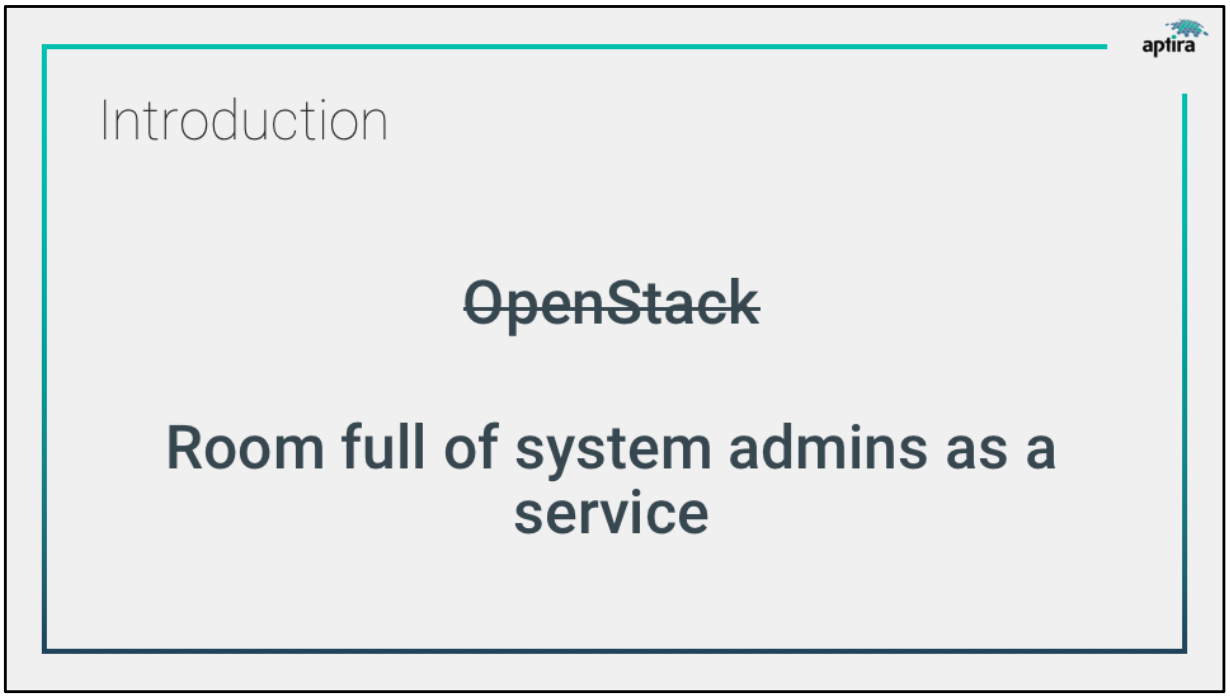
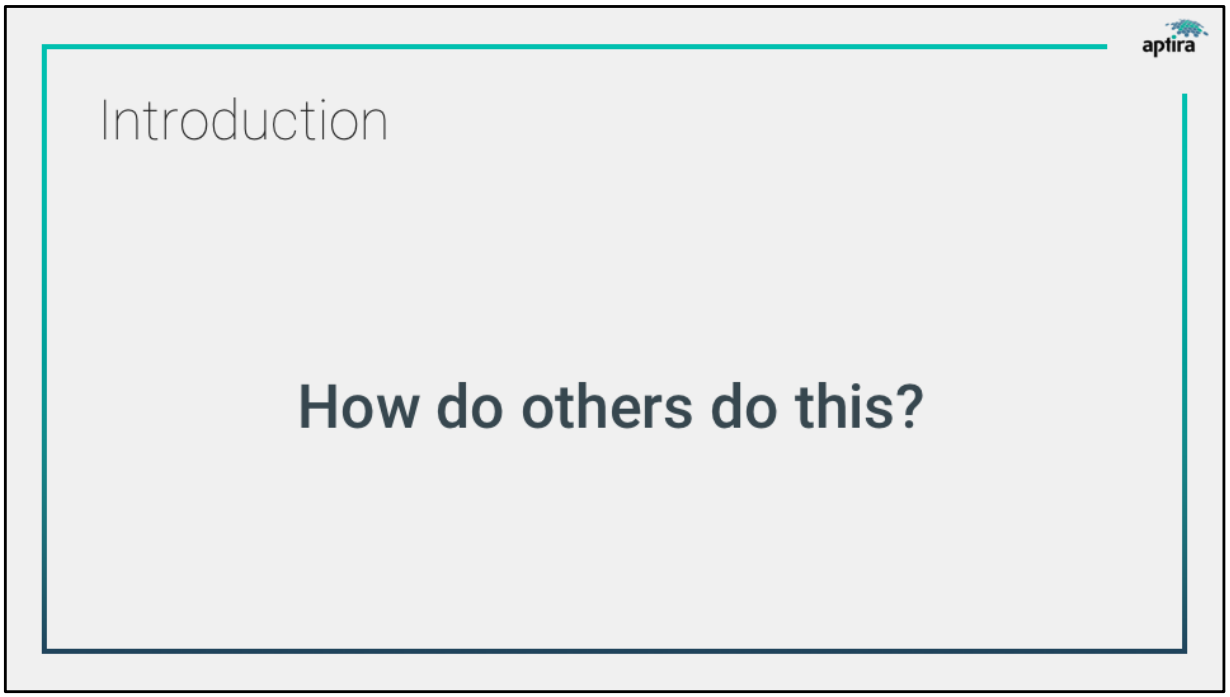
Who am I? I'm a python developer who has been working on OpenStack since 2011. I currently work for Aptira, who do OpenStack, SDN, and orchestration consulting. I'm here today to help you learn from my fail.



OpenStack is an orchestration system for setting up virtual machines and associated other virtual resources such as networks and storage on clusters of computers. At a high level, OpenStack is just configuring existing facilities of the host operating system -- there isn't really a lot of difference between OpenStack and a room full of system admins frantically resolving tickets requesting virtual machines be setup. The only real difference is scale and predictability.

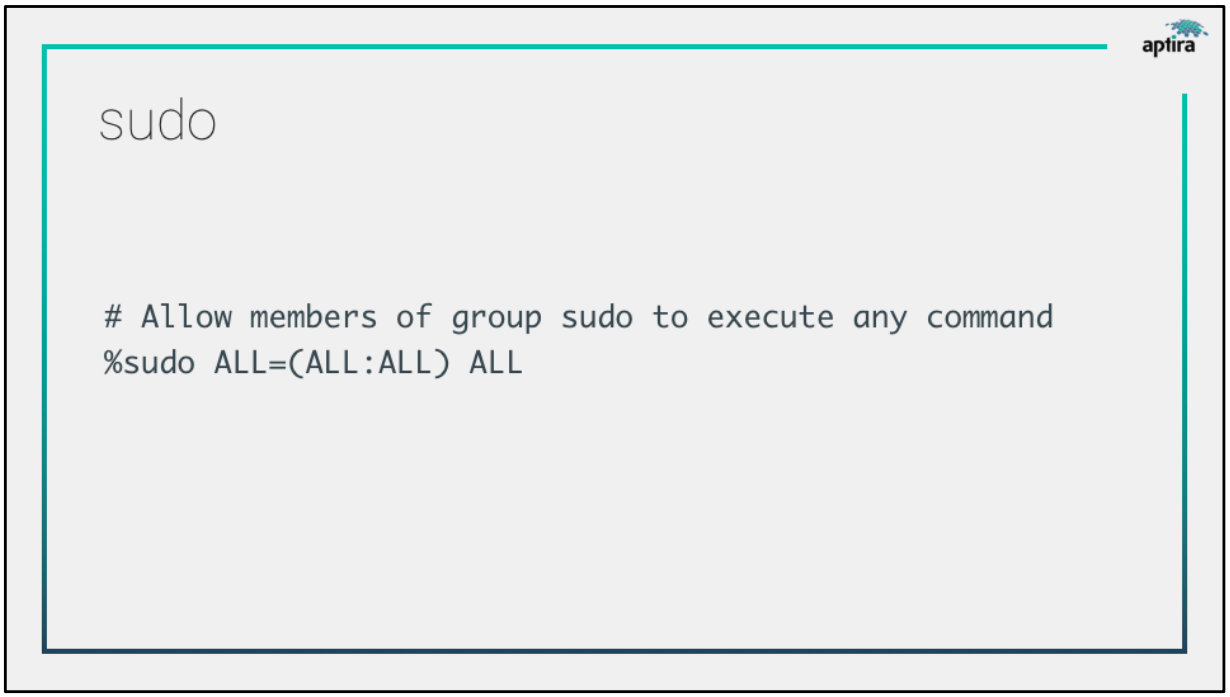
To do its job, OpenStack needs to be able to manipulate parts of the operating system which are normally reserved for administrative users. This talk is the story of how OpenStack has done that thing over time, what we learnt along the way, and what I'd do differently if I had my time again. Lots of systems need to do these things, so even if you never use OpenStack hopefully there are things to be learnt here.

That said, someone I respect suggested last weekend that good conference talks are actionable. A talk full of OpenStack war stories isn't actionable, so I've spent the last week re-writing this talk to hopefully be more of a call to action than just an interesting story. I apologise for any mismatch between the original proposal and what I present here that might therefore exist.



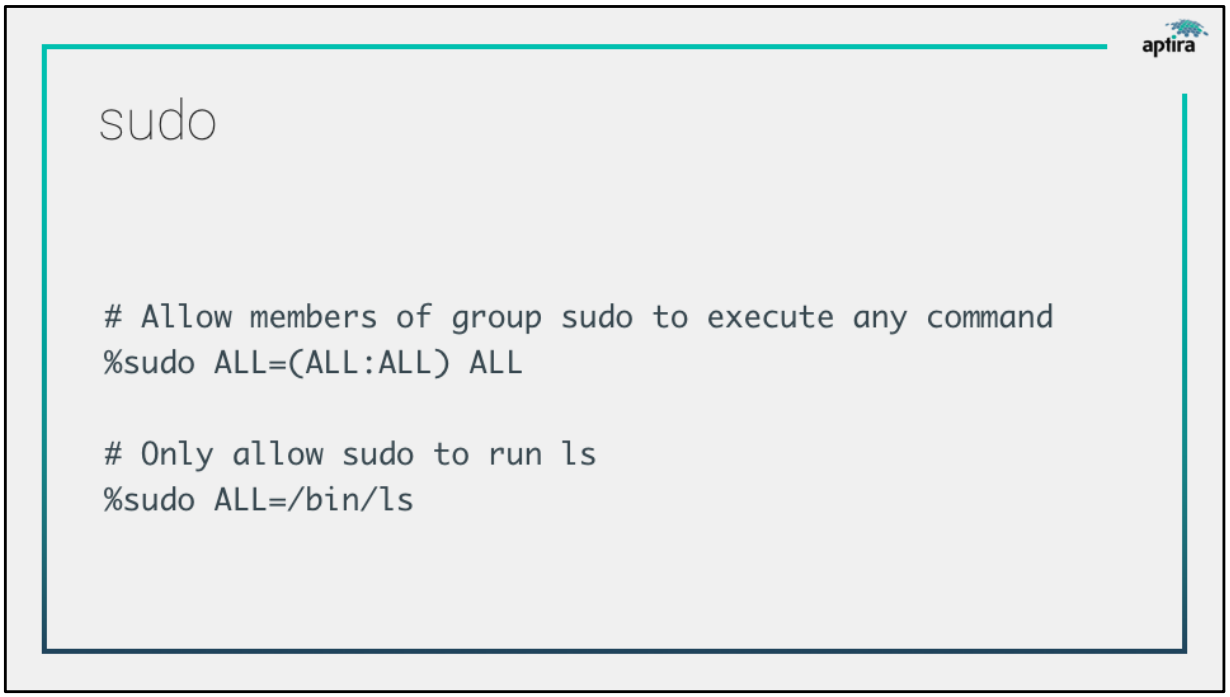
Back to the task in hand though -- providing control of virtual resources to untrusted users. OpenStack has gone through several iterations of how it thinks this should be done, so perhaps its illustrative to start by asking how other similar systems achieve this. There are lots of systems that have a requirement to configure privileged parts of the host operating system. The most obvious example I can think of is Docker. How does Docker do this? Well... its actually not all that pretty. Docker presents its API over a unix domain socket by default in order to limit control to local users (you can of course configure this). So to provide access to Docker, you add users to the docker group, which owns that domain socket. The Docker documentation warns that "the docker group grants privileges equivalent to the root user". So that went well.

Docker is really an example of the simplest way of solving this problem -- by not solving it at all. That works well enough for systems where you can tightly control the users who need access to those privileged operations -- in Docker's case by making them have an account in the right group on the system and logging in locally. However, OpenStack's whole point is to let untrusted remote users create virtual machines, so we're going to have to do better than that.

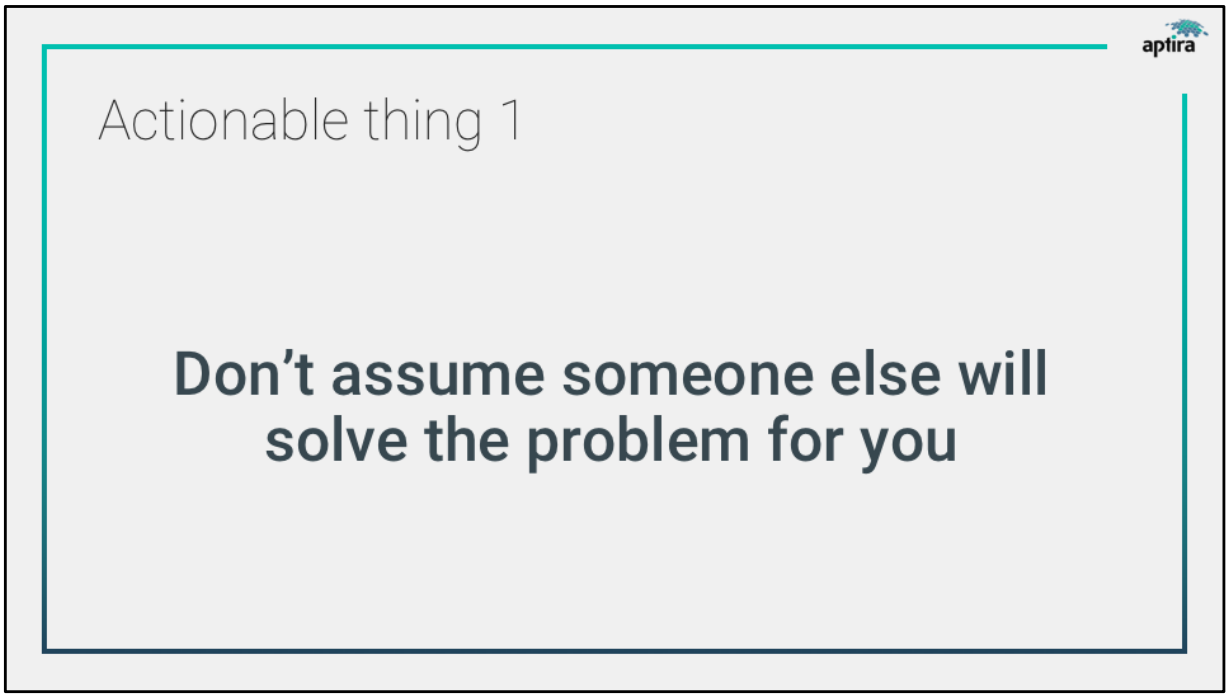


The next level up is to do something with sudo. The way we all use sudo day to day, you allow users in the sudoers group to become root and execute any old command, with a configuration entry that probably looks a little like this.

Now that config entry is basically line noise, but it says "allow members of the group called sudo, on any host, to run any command as root". You can of course embed this into your python code using subprocess.call() or similar.



On the security front, its possible to do a little bit better than a "nova can execute anything" entry. For example, this says that the sudo group on all hosts can execute /bin/ls with any arguments. OpenStack never actually specified the complete list of commands it executed. That was left as a job for packagers, which of course meant it wasn't done well.

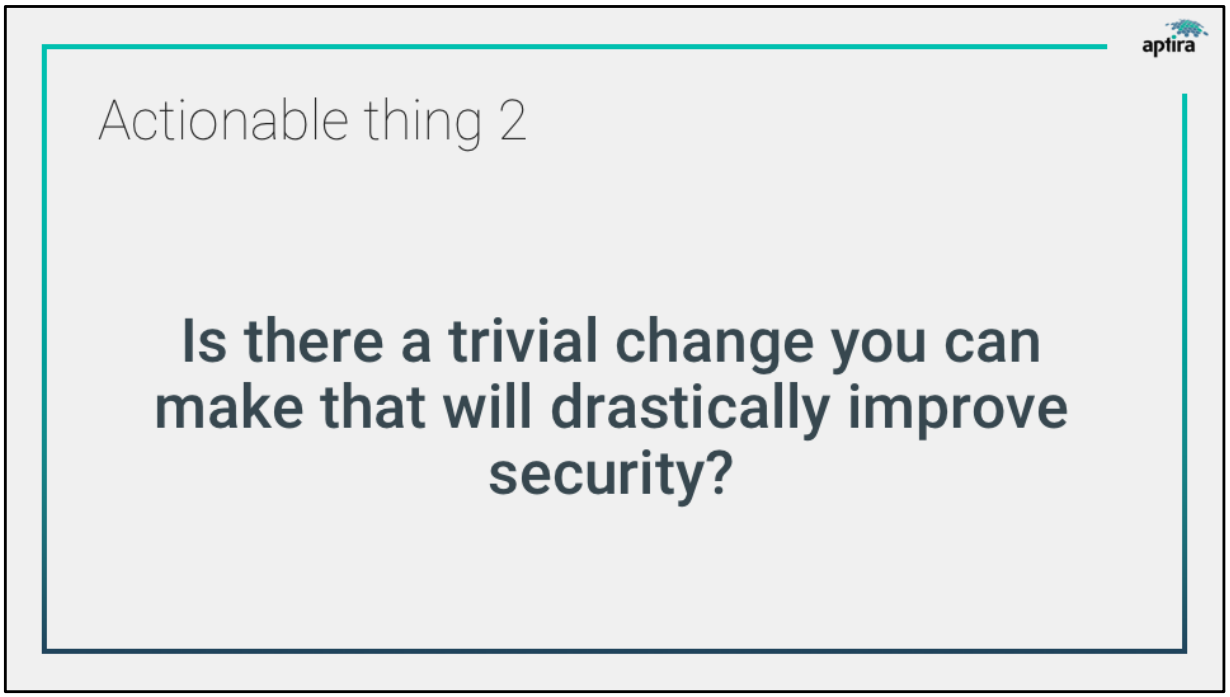


So there's our first actionable thing -- if you assume that someone else (packagers, the ops team, whoever) is going to analyse your code well enough to solve the security problem that you can't be bothered solving, then you have a problem. Now, we weren't necessarily deliberately punting here. Its obvious to me how to grep the code for commands run as root to add them to a sudo configuration file, but that's unfair. I wrote some of this code, I am much closer to it than a system admin who just wants to get the thing deployed.

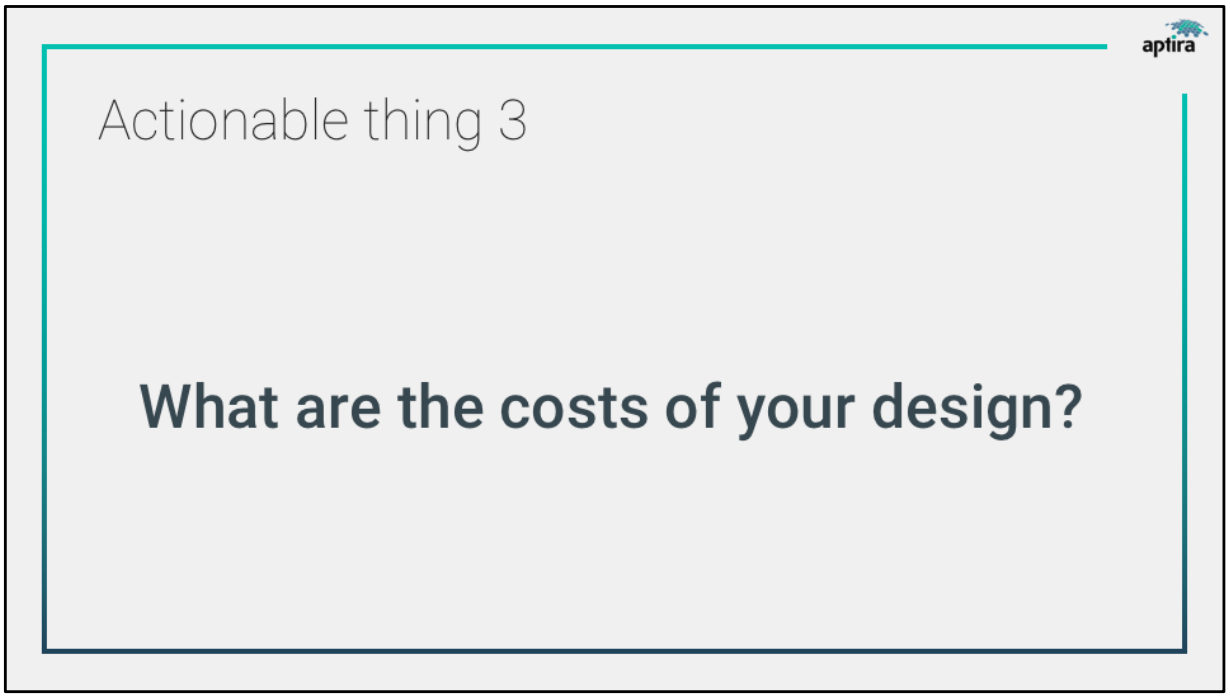
We can of course do better than just raw sudo. Next we tried a thing called rootwrap, which was mostly an attempt to provide a better boundary around exactly what commands you can expect an OpenStack binary to execute. So for example, maybe its ok for me to read the contents of a configuration file specific to a virtual machine I am managing, but I probably shouldn't be able to read /etc/shadow or whatever. We can do that by doing something like the example shown where, where nova-rootwrap is a program which takes a configuration file and a command line to run. The contents of the configuration file are used to determine if the command line should be executed.

Now we can limit the sudo configuration file to only needing to be able to execute nova-rootwrap.

I thought about putting in a whole bunch of slides about exactly how to configure rootwrap, but then I realised that this talk is only 25 minutes and you can totally google that stuff.

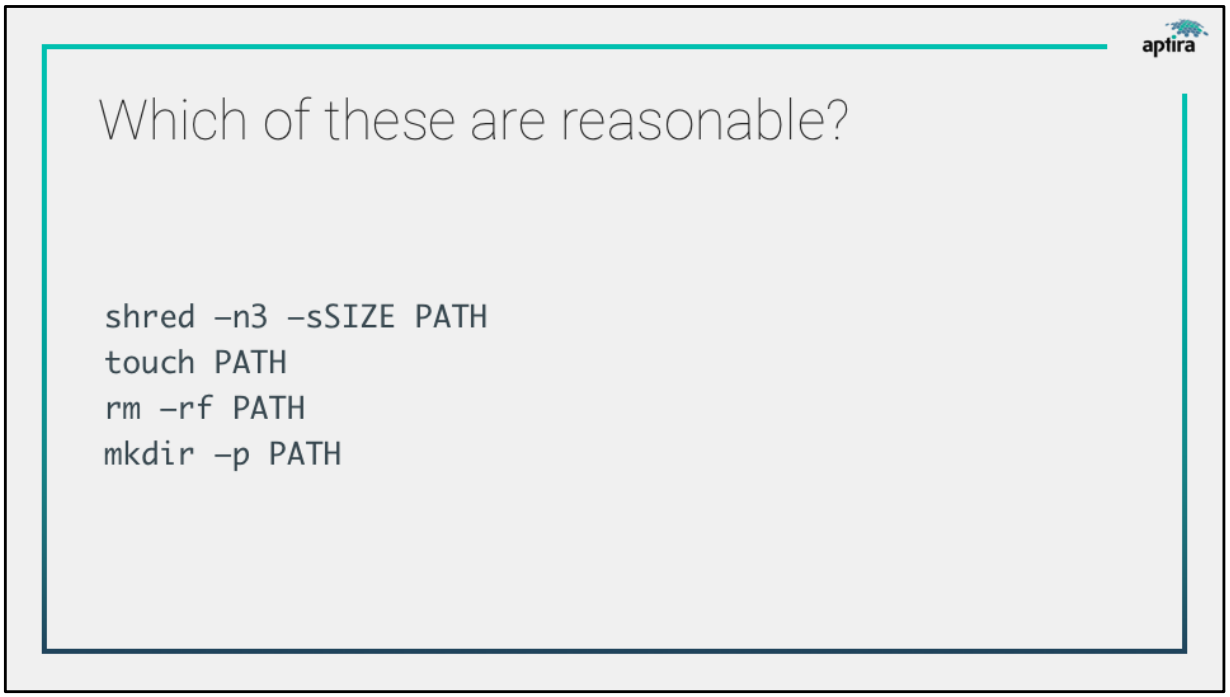


So instead, here's my second actionable thing... Is there a trivial change you can make which will dramatically improve security? I don't think anyone would claim that rootwrap is rocket science, but it improved things a lot -- deployers didn't need to grep out the command lines we executed any more, and we could do things like specify what paths we were allowed to do things in. Are there similarly trivial changes that you can make to improve your world?



But wait! Here's my third actionable thing as well -- what are the costs of your design? Some of these are obvious -- for example with this design executing something with escalated permissions causes us to pay to fork a process. In fact its worse with rootwrap, because we pay to fork, start a python interpreter to parse a configuration file, and then fork again for the actual binary we wanted in the first place. That cost adds up if you need to execute many small commands, for example when plugging in a new virtual network interface. At one point we measured this for network interfaces and the costs were in the tens of seconds per interface.

There is another cost though which I think is actually more important. The only way we have with this mechanism to do something with escalated permissions is to execute it as a separate process. This is a horrible interface and forces us to do some really weird things. Let's checkout some examples...



Which of the following commands are reasonable?

These are just some examples, there are many others. The first is probably the most reasonable. It doesn't seem wise to me for us to implement our own data shredding code, so using a system command for that seems reasonable. The other examples are perhaps less reasonable -- the rm one is particularly scary to me. But none of these are the best example...

How about this one?

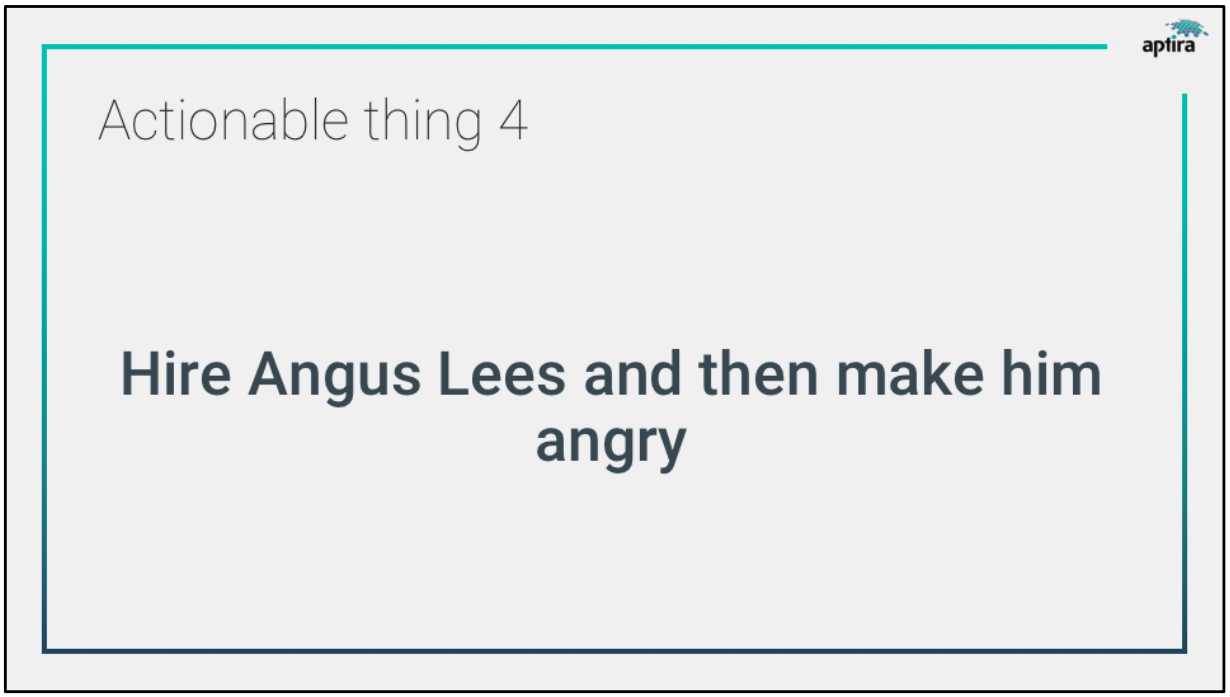
Some commentary first. This code existed in the middle of a method that does other things. Its one of five command lines that method executes. What does it do?

Its actually not too bad. Using root permissions, it writes a zero to the multicast_snooping sysctl for the network bridge being setup. It then checks the exit code and raises an exception if its not 0 or 1.

That said, its also horrid. In order to write a single byte to a sysctl as root, we are forced to fork, start a python process, read a configuration file, and then fork again. For an operation that in some situations might need to happen hundreds of times for OpenStack to restart on a node.

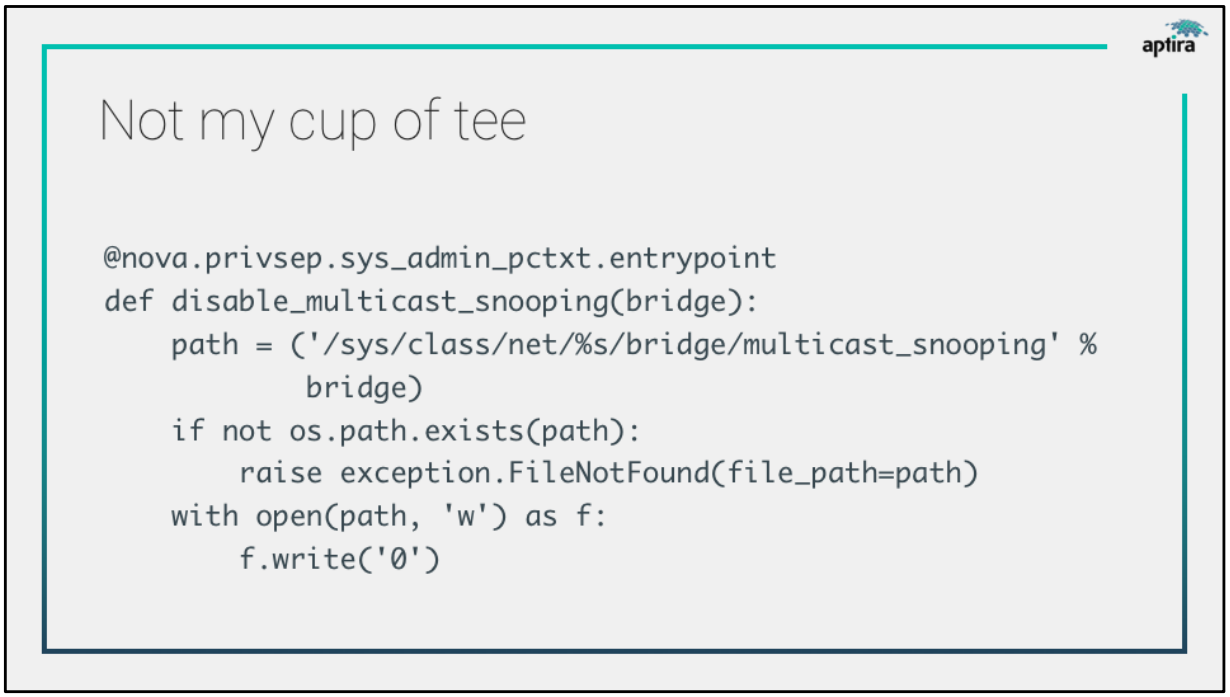
This is how we get to the third way that OpenStack does escalated permissions. If we could just write python code that ran as root, we could write this instead.

Its not perfect, but its a lot cheaper to execute and we could put it in a method with a helpful name like "disable multicast snooping" for extra credit. Which brings us to...



Hire Angus Lees and make him angry. Angus noticed this problem well before the rest of us. We were all lounging around basking in our own general cleverness. What Angus proposed is that instead of all this forking and parsing and general mucking around, that we just start a separate process as at startup with special permissions, and then send it commands to execute.

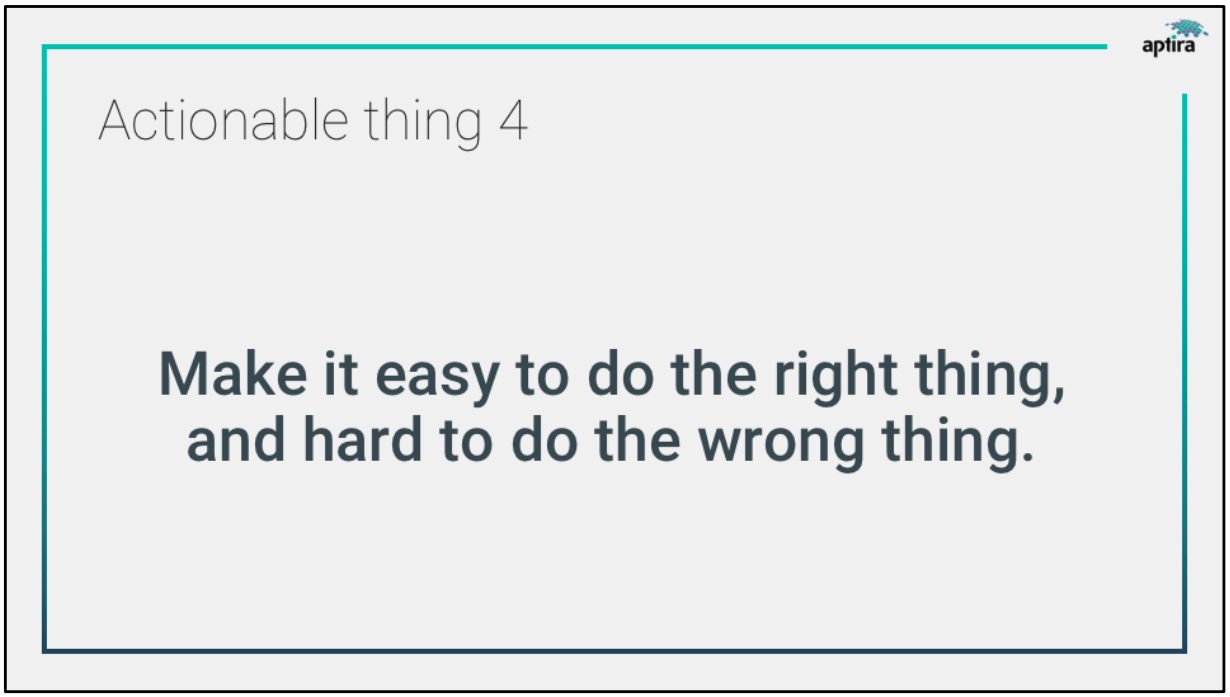
He could have done that with a relatively horrible API, for example just sending command lines down the pipe and getting their responses back to parse, but instead he implemented a system of python decorators which let us call a method which is marked up as saying "I want to run as root!".



So here's the destination in our journey, how we actually do that thing in OpenStack now.

The decorator before the method definition is a bit opaque, but basically says "run this thing as root", and the rest is a method which can be called from anywhere within our code.

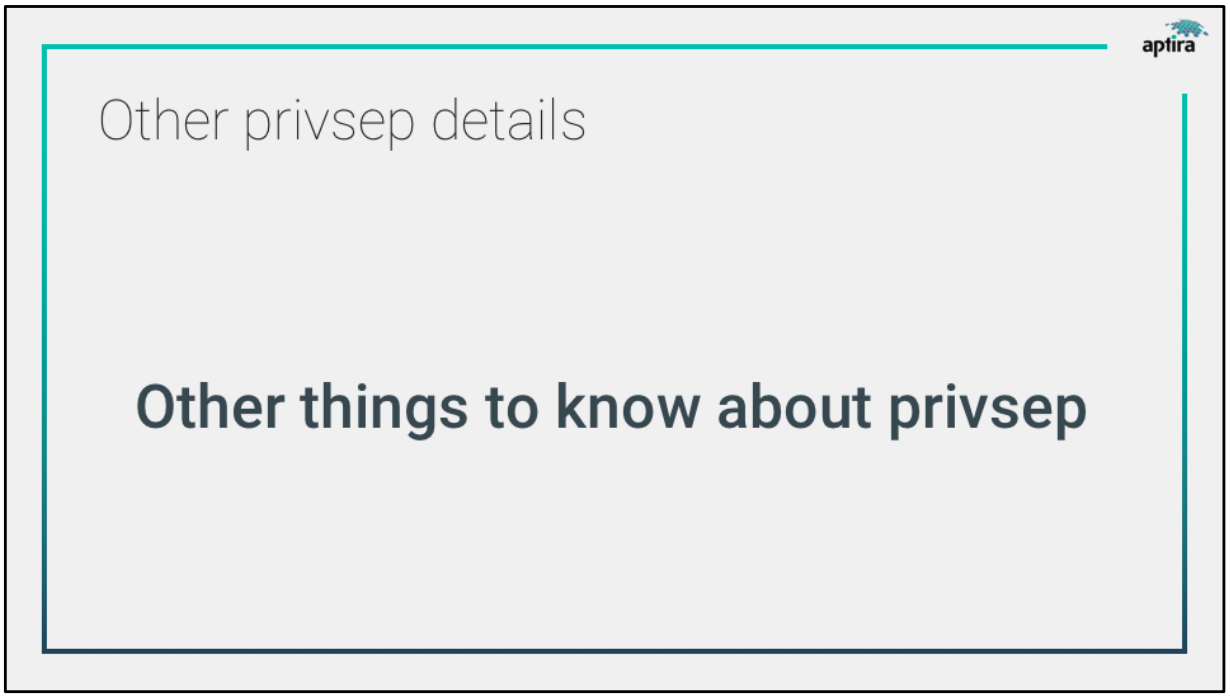
There are a few things you need to do to setup privsep, but I don't have time in this talk to discuss the specifics. Effectively you need to arrange for the privsep helper to start with escalated permissions, and you need to move the code which will run with one of these decorators to a sub path of your source tree to stop other code from accidentally being escalated. privsep is also capable of running with more than one set of permissions -- it will start a helper for each set. That's what this decorator is doing, specifying what permissions we need for this method.



And here we land at my final actionable thing. Make it easy to do the right thing, and hard to do the wrong thing. Rusty Russell used to talk about this at linux.conf.au when he was going through a phase of trying to clean up kernel APIs -- its important that your interfaces make it obvious how to use them correctly, and make it hard to use them incorrectly.

In the example used for this talk, having command lines executed as root meant that the prevalent example of how to do many things was a command line. So people started doing that even when they didn't need escalated permissions -- for example calling mkdir instead of using our helper function to recursively make a set of directories.

We've cleaned that up, but we've also made it much much harder to just drop a command line into our code base to run as root, which will hopefully stop some of this problem re-occuring in the future. I don't think OpenStack has reached perfection in this regard yet, but we continue to improve a little each day and that's probably all we can hope for.



privsep can be used for non-OpenStack projects too. There's really nothing specific about most of OpenStack's underlying libraries in fact, and there's probably things there which are useful to you. In fact the real problem is working out what is where because there's so much of it.

One final thing -- privsep makes it possible to specify the exact permissions needed to do something. For example, setting up a network bridge probably doesn't need "read everything on the filesystem" permissions. We originally did that, but stepped back to using a singled escalated permissions set that maps to what you get with sudo, because working out what permissions a single operation needed was actually quite hard. We were trying to lower the barrier for entry for doing things the right way. I don't think I really have time to dig into that much more here, but I'd be happy to chat about it sometime this weekend or on the Internet later.

aptira

Summary

So here's a quick summary of my actionable points for this talk:

- Don't assume someone else will solve the problem for you.
- Are there trivial changes you can make that will drastically improve security?
- Think about the costs of your design.
- Hire smart people and let them be annoyed about things that have always "just been than way". Let them fix those things.
- Make it easy to do things the right way and hard to do things the wrong way.

aptira

And a final note

I really struggled with writing these slides, so in the end I wrote this talk up as a blog post first and then turned it into slides. You can find the blog post version of this talk at:

http://madebymikal.com/python/

I'd be happy to help people get privsep into their code, and its very usable outside of OpenStack. There are a couple of blog posts about that on my site:

http://www.madebymikal.com/?s=privsep

But feel free to contact me at mikal@stillhq.com if you'd like to chat.


aptira
https://aptira.com
Australia freecall: 1800 APTIRA
International: +612 8030 2333
Follow: @aptira