

# Python Plus osquery

Today I'm giving an introduction to osquery in general and accessing it from python in specific, since osquery is a versatile enough tool that you can interact with it in multiple ways:

Allister Banks

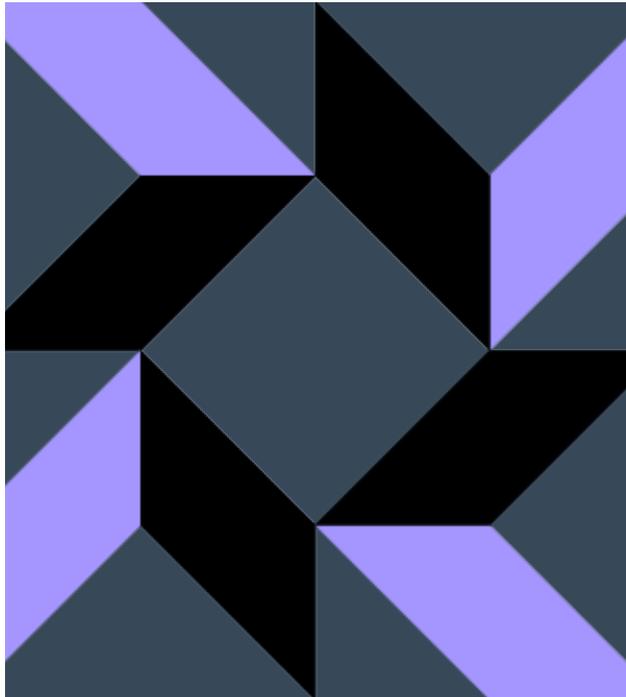
---

@allister on the MacAdmins Slack,

---

[github.com/arubdesu](https://github.com/arubdesu)

I'd be silly not to mention logging right off the bat as the primary recommended way to use it at scale, so I'll touch on that briefly, but you can also run ad-hoc commands in terminal or from scripts and do some reporting with things like JAMF's casper suite or the Sal dashboard for munki. osqueryi is the interactive shell you can use when trying to figure out what data you want to pull out or correlations you want to make.

The logo for osquery, featuring a stylized geometric pattern of triangles in shades of purple, black, and grey, arranged in a diamond-like shape.

# osquery

- Logging
- Ad-Hoc via osqueryi
- via the Python module

And it may be a little convoluted, but I wanted to show how I created an adhoc auditing tool via the python module that they maintain to generate a customized inventory for Albert Einstein College of Medicine. My employer Montefiore Medical Group owns a whole bunch of hospitals from the Bronx to southern Westchester in NY, and we run data centers for other hospitals but are only now fully taking on support for the college, which we've had a long affiliation with.

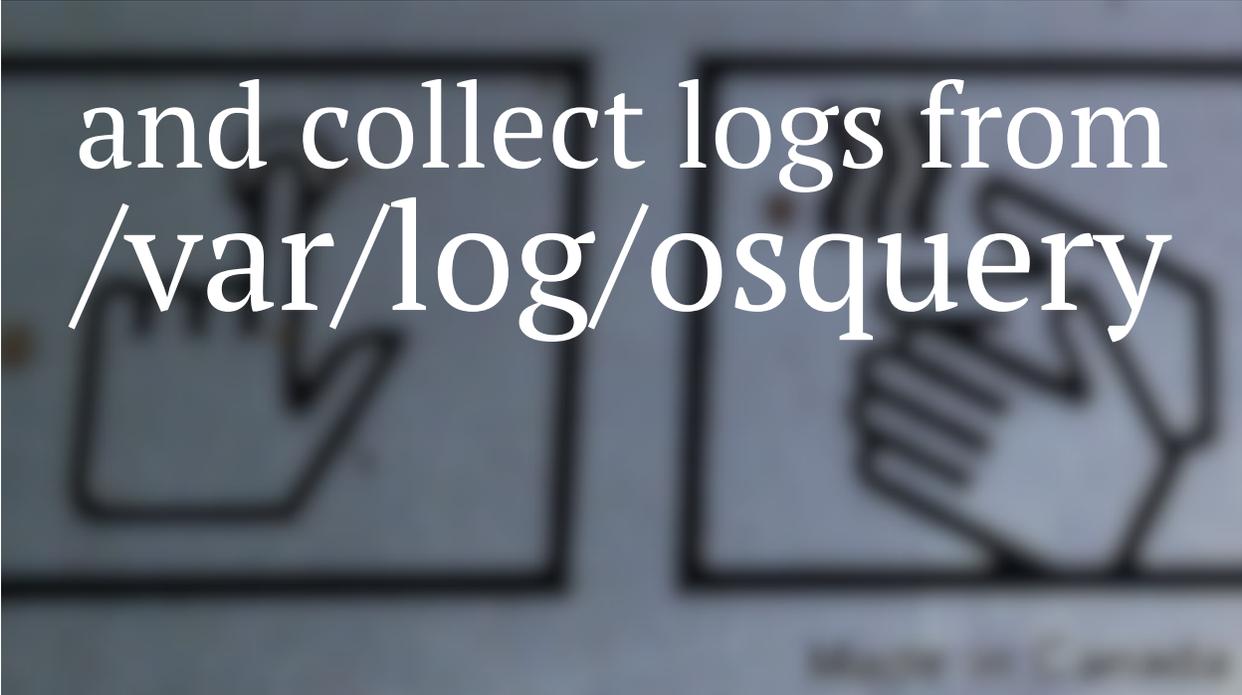
I'd like to fast-forward to the good stuff, or the cliff notes or spark notes version of the talk, borrowing Ed Marczak's line from all of his presentations why do you want to use osquery?

# Because system\_profiler is slooow

How do I get this thing and make it run? Thanks to sam keeley, between this autopkg run and the included osqueryctl command,

```
autopkg install osquery  
sudo osqueryctl start
```

you can immediately start a daemon that runs one query every hour for the hostname, cpu and physical memory of the computer to a results log in /var/log/osquery, which you can then expand on as you get the hang of it,

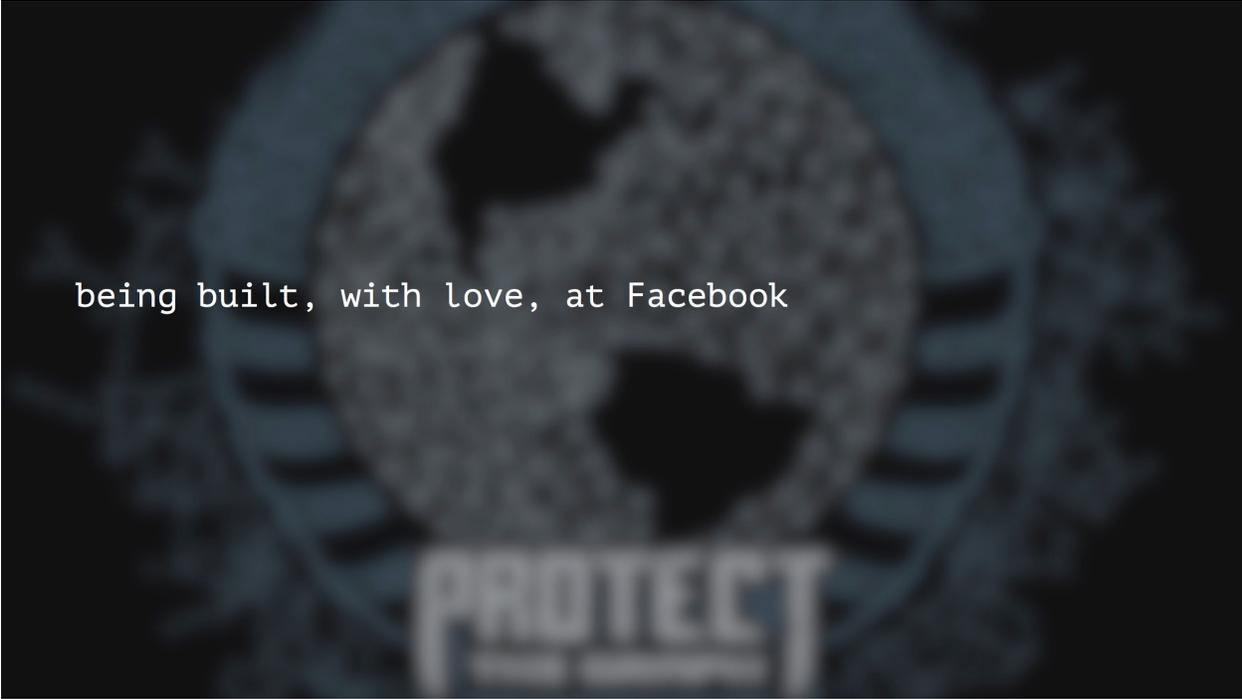


and collect logs from  
/var/log/osquery

# Or,

[github.com/arubdesu/EAs](https://github.com/arubdesu/EAs)

Or, if you don't want to think about managing or parsing logs in general, I did wire up an adhoc method of running it that you can find in that [github repo](#), I recommend auditing the [launchd\\_check](#) python script in particular.



being built, with love, at Facebook

Facebook released osquery as open source in late 2014, and it really landed on my radar around the MacBrained session in February 2015 presented by Mike Arpaia, (who is now the manager of that team at Facebook but made commits as recently as yesterday), and you can find that session archived on the MacBrained Live ustream video page.

The only time I had paid attention to osquery before then was when it hit the social media with an indication that it would be something special for Macs. But the message to the world at large was that you'd instrument or pull data from a Mac by actually making database queries in sql, and in the hangover from cfprefsd wrecking our file-based pref mgmt ways I kindof think everyone was like 'thanks, but no thanks'.

# PSU MacAdmins 2015



Teddy Reed joined Mike Arpaia to present on osquery at the PSU MacAdmins conf last year, so I should be able to point to that as 'prior art', but it was a bit more focused on them as having these specialized host intrusion detection needs. There is, of course, a lot of value in centralized logging in general, and I'm going to 'go long' as it were on that application of the tool at MacDevOps YVR in June, right before PSU, so hopefully you don't mind if just I highlight some marquee features and concepts we'll need for the purpose of my destination for this talk.



For a brief exposition on why the folks at Facebook wrote it, and background as to some other devops-y type tools that are in this space: I'd say one of the few popular precursors to osquery is the company formerly known as Puppetlabs facter(I keep going 'Puppet the company' now that they changed their company name AGAIN recently)



facter, like puppet, was originally written in ruby and is purpose-built to inform configuration management decisions. If you're familiar with puppet you may also know there is so much you can express in puppet as a language now and Chef has an even more native-ruby-ish way of writing configs that it's referred to as a DSL - or domain specific language, which just means a purpose-built way to express things.

Deployments like Google's Mac fleet use facter extensively to do certain things for some of their machines, based on the criteria that's returned, just in the same way a smartGroup in the Casper suite would allow the application of policies.

At Facebook they don't use puppet, and while they probably have 50,000 less Macs than Google has, they wanted osquery to be a lot more efficient and native than Ruby can be, since Apple has been screwing with folks trying to access system frameworks from Ruby for quite some time.

I also hear comparisons drawn to a tool from yelp called osxcollector, which is in python, but it's more to generate forensics data to be used after a machine has had malware or other types of funny business experienced, and you'd refer to it in the class of digital forensics and incident response tools.

The



osquery developers have extensive python development experience, but there's a history of security tools that they helped develop that didn't do everything they had hoped for and didn't quite take off, so this time around their goals were clearly stated:

- They wanted easy enough for people from other parts of IT to collaborate on,
  - They wanted fast and stable, so they could run this on actual production Linux server instances, as they are doing so right now around the world
- and also they needed to know it wouldn't either interfere with or be messed up by being on developers laptops.

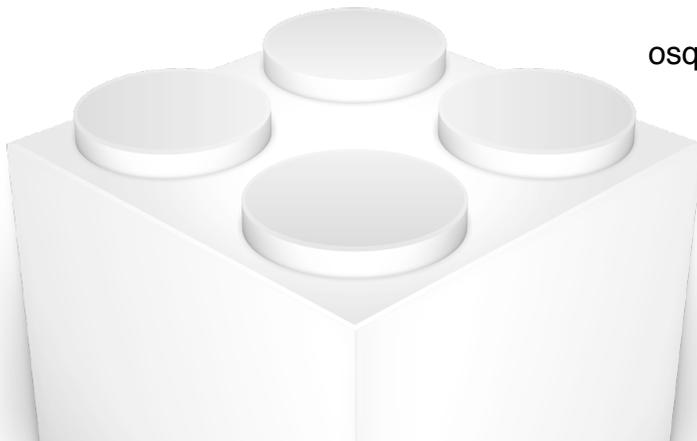


from the very first [introduction video](#), lovingly stolen from [Mike Arpaia's slides](#), 2014

So as languages go, they refer to what osquery is written in as objectiveC++. They wanted to assume that, if you're listening in the least disruptive and minimally resource-intensive manner possible, you can listen for more things and the tale of the tape of what you were interested in should give you an indication of why things happened BEFORE you need an incident response-type situation.



And one last lecture-y point before I get into real-world usage that I recommend folks start with - I spoke of a DSL in terms of puppet and chef, well they chose sqlite3 as the DSL for getting the data in via the queries you're essentially running locally on each machine, and I believe the motivation for that choice is it's pretty standard to use regex or some other query-type format to get data out after you've done the log aggregation and sql is just slightly less obscure and brain-breaking. The osquery devs contribute to the sqlite3 spec and arrange the events it can listen to or information you'd be interested in keeping tabs on in special tables or groupings of data.



osquery's "killer demo", in my opinion, is its lightning-fast retrieval of loaded kernel extensions, versus the 20+ seconds system\_profiler takes to return the same info. osquery itself has a kernel extension, which isn't something you should think of as so fancy, you might more commonly refer to software like this as drivers.

Google's santa project uses one, and another security-related kernel extension is by a company and developer well-known in Apple infosec circles, Stefan Esser (that's his github avatar below) of SektionEins in Germany, released a kext called SUIDGuard. If you remember last years talk by Jesse Shipley, it was released as a mitigation against an exploit for Yosemite.



Well it turns out one-off code that runs at the kernel level, especially when aimed at something Apple wants to target with their own fixes is actually risky, and caused issues on 10.11.4 upgrades because it wasn't even warrantee'd to run on anything other than 10.10 anyway. From a stability perspective, if any of your customers that have admin installed it you'd probably want to know even before they upgraded to ElCap that it was there. As I understand it, Apple has since stopped old versions from being loaded and causing issues during boot, but this is one of those things that folks should really keep an eye on.

Apple has required signed Kernel extensions as of ElCap because it's really trivial to get around security frameworks like SIP if you're at that low kernel level. It's exactly what Pedro Vilaça's rootfool software is meant to perform - SIP unlock without as much as a reboot.

## Rootfool – a small tool to dynamically disable and enable SIP in El Capitan

🕒 October 12, 2015 📁 Tools

El Capitan is finally released and System Integrity Protection aka SIP aka rootless is finally a reality we must face. Let me briefly describe SIP (technical details maybe in another post, now that El Capitan is final and out of NDAs). This [post](#) by Rich Trouton contains a very good description of its userland implementation and configuration.

*What is SIP anyway?*

The description that I like to use is that SIP is a giant system-wide sandbox, that controls access to what Apple considers critical files and folders. One of the reasons for this is that most of kernel side SIP implementation exists into the `Sandbox.kext`, the same TrustedBSD kernel extensions that implements OS X sandbox mechanism.

As I demo'd you try running `system_profiler` to get info on whether these kexts are installed or loaded, go get a cup of coffee because it's going to take a while, and if you want to collect that at scale you're slowing everything down, versus having a purpose-build tool collect that info for you.

A further extension of what makes `osquery` awsm is the concept of an event stream which is pretty basic, it just means that if your management tools only check state in a preflight or postflight you wouldn't catch any updates that happened in the meantime. It's a common computer science concept to have something publishing or pushing out data, and have a 'subscriber', kindof like to your youtube channel about unpacking scented candles or like an rss feed, so that you don't keep having to going back and check if there's a new episode.



To show what classes of data osquery can subscribe to, or put another way, generate logged events in a stream that buffer and then write to logs when the system isn't busy doing other stuff, let's go to the osquery [general-public-facing project page](#). Everything you see here with a database-type icon across from the table name is that event stream type, including if a file gets changed, or if malware has downloaded other software and mounted it as a dmg. The rule of thumb is if the name ends in 'events' then it's that you'll always get the breaking news updates, event stream type.

One last thing before I move on from the basics of osquery, and that is people might think to themselves 'I'm not a DBA or database admin, nor in infosec, which end do I blow in?' or 'how would I build up queries of things worth auditing'. Hopefully my casper extension attribute repo on github gives you some ideas, but there's a larger osquery community that has you covered from that perspective as well, via a concept called '[packs](#)', or collections grouped by use case. If you're more concerned about potential vulnerabilities rather than attacks, there's a pack for that. If you have a lot of admins and need to check basic controls are in place there's an IT compliance pack, and with the addition of Windows in a release dropping Any Day Soon Now™, you'll be glad to notice that you can make your own packs and distribute them and have them apply optionally as you see fit.



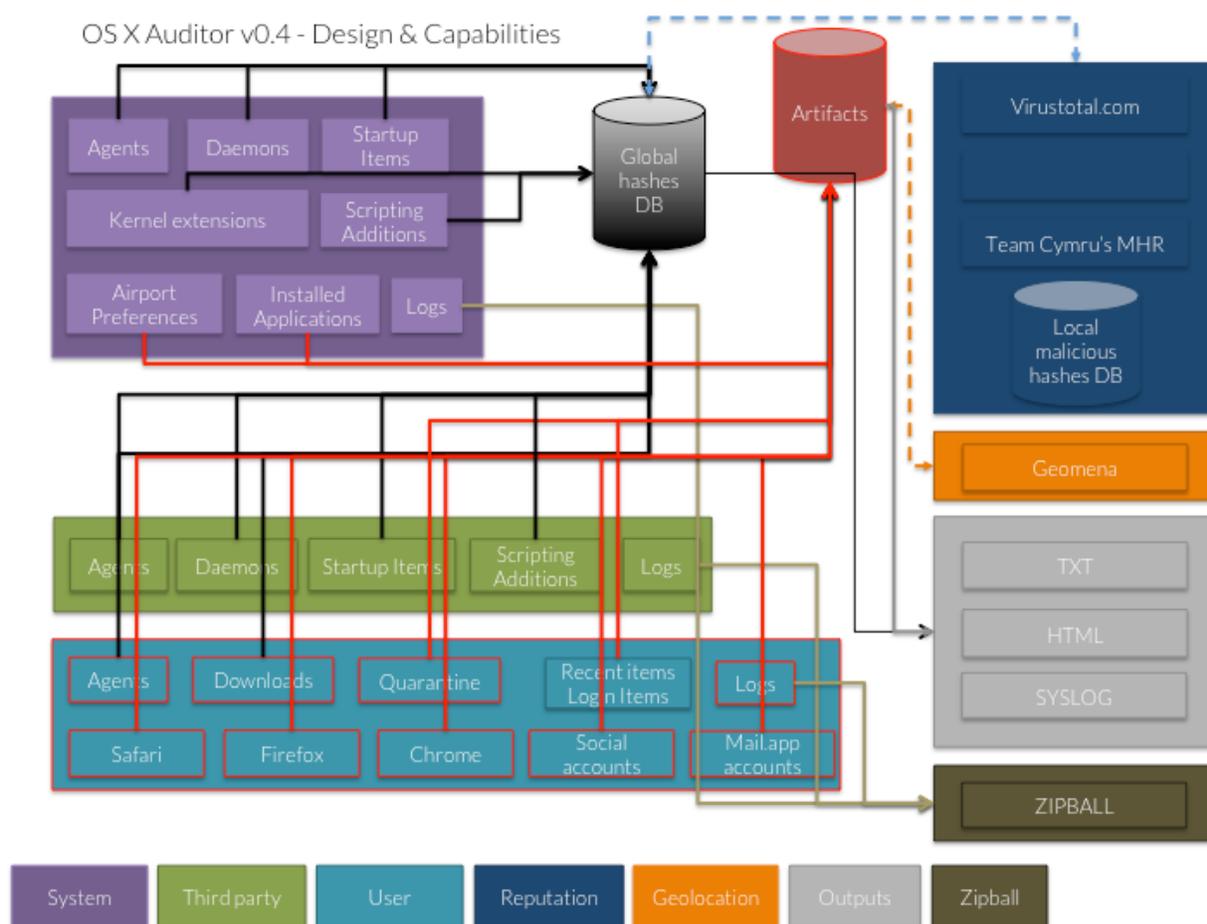
But, while we're on that point, we've come to that special part in many presentations, which starts with 'If you take away nothing else from this talk...'

In this instance I'm going to cap that off with 'the top two things to know, about the systems under your care are

1. what has persistence and
2. what browser extensions are installed.'

You get a good handle on that, or keep it in your field of vision, and you've covered the lion's share of anything that would bite you. I'm going to get back to that first point of stuff like daemons that stay loaded, but the majority of complaints customers have are going to be browser-related, so I'll go over my evolution with auditing browser extensions on my fleet.

If you've interacted with me in, for example, the #bash channel in MacAdmins Slack, you'll notice I get very impatient with anything complex that people are trying to do with bash, which usually means anything other than simple one-liners or the most basic loops, because I prefer python.



And it came in handy a bunch of times, like when I first looked into grabbing all browser extensions on a system to whitelist or flag with a precursor tool to osxcollector called osxauditor.

It wasn't working for all browsers when I tested it, but between code I found in it and poking around I was able to build a process that works on 10.9 and greater machines to list some metadata about all of the browser extensions present on a system, whether or not they're actually active, and put that in an EA with basic prettified output.

I then put in a feature request and a few weeks later a stable version of osquery was shipping with an approximation of what I had done mostly manually with python. Since I had already written my version, which includes the whitelist I decided upon for the ones I felt were safe or appropriate enough for my environment, I'm still using the version you see in my EAs repo on github for Casper in prod, but many of my more recent ones include osquery output as the way I derive if there is info of interest on a machine. It wasn't long before I'd built up a pretty good repertoire, and exposed the limits that not using a log aggregator causes. And just to show you some of the why and how...

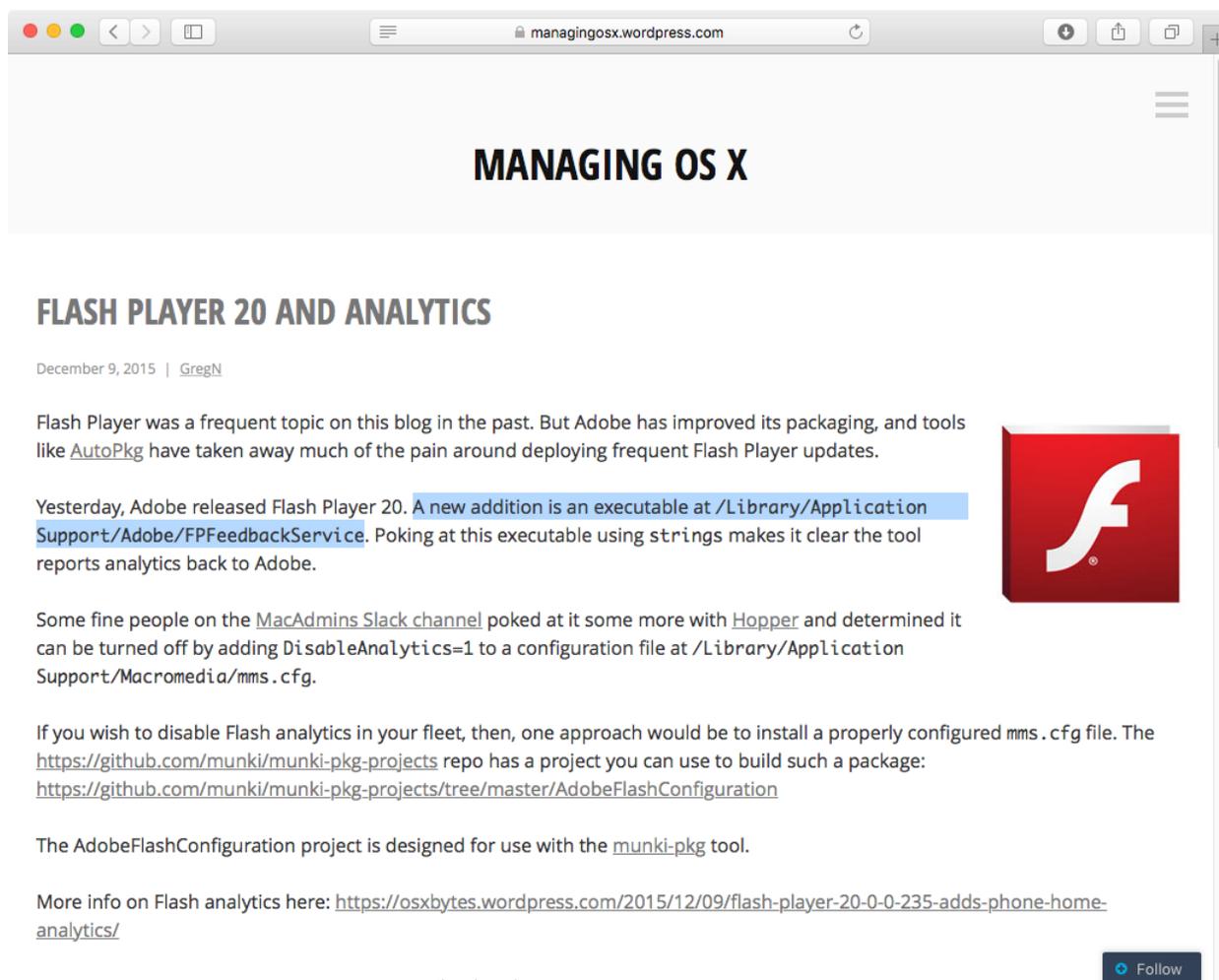
This is an incarnation I wrote for fun over the weekend - separate out the whilelist data, start with a check for if either osquery and/or the python module is actually installed, open a osqueryd process once and shove a query for each of the browsers on my system, and then pour out the rest of the metadata osquery collects if it finds one that isn't in the whitelist.

And finally for that auditing project, customers at the university downloaded an app bundle that I hacked together between applescript for just a native app look that would definitely have OS compatibility way back to 10.5 and a bundled version of cocoadialog so I could get user input for one piece of information, their email address, via Google's gmacpyutility interface to cocoadialog. We then called open from the command line with a URL to send them to a web form where they'd tell us building, floor, room and department because the network team was unwilling or unable to correlate that for us, and while they were filling that form out I was opening a bundled version of osqueryd to shove a bunch of queries through all at once to keep it very lightweight and fast, then dumping them to a csv and curl'ing it to Box.com's anonymous file upload for further parsing.

# There's more...

Now's about the time I should be wrapping up and saying thank yous and references, but I said that there's two things to take away from this talk, and besides keeping browsers clean it is to watch for persistence which means things like cron jobs, rc scripts, and if you just browse those packs on the osquery.io site you'll notice the majority of them, (that are about protecting Macs) all look at launchd jobs.

When I was relying on the JAMF Casper suite for inventory and really only learning about things like pylint to make my python a bit more readable to other people in the community, I started using osquery to watch launchd, and after a bit of open monitoring, collected a list of things that I could have a reasonable sense of confidence weren't malware. I realize it's kind of the Godwin's law of macadminery to discuss Adobe, but...



The screenshot shows a browser window with the URL `managingosx.wordpress.com`. The page title is **MANAGING OS X**. Below it is a sub-header **FLASH PLAYER 20 AND ANALYTICS**. The post is dated December 9, 2015, by GregN. The main text discusses Adobe's Flash Player 20, mentioning that it includes analytics reporting back to Adobe. A red square with a white 'f' logo (the Adobe Flash logo) is positioned to the right of the text. The text includes several links to GitHub repositories and a previous blog post. At the bottom right of the post content, there is a 'Follow' button.

Adobe released flash player 20.0.0.235 with an analytics service binary and a new launchd job that auto-updates would apply and start running silently. 70+ CVE's security bulletins were addressed by the patch part of that release, but I pulled it as soon as the canaries in my munki release process sent in reports that knew unknown services were running because of it. Autopkg loaded it in and I committed it, so it's not like I wasn't aware of where it came from, but having osquery report on all launchd jobs including user-level launchagents meant I saw it as soon as it was present on-disk. In the autopkg community we had already prioritized checking downloaded packages or apps for the developers signatures, but verifying that something malignant hasn't been slipped into the build or release process relies on looking at behavior.

# Go forth and query

I hope that I've convinced you to start to find a way to check for those two groups of data, browser extensions and launchd jobs, and shown off some ways that incorporating osquery into my sysadmin tasks has really paid off. And now if folks don't have any questions we can just tool around in the interactive session or go over docs, or the contribution process of how they happen to run the project.

# Thanks!

## Python Plus [osquery](#), by Allister Banks

---

Created for Philly MacAdmins, April 21, 2016

## Introduction, TL;DR

---

*What's this about / what's the quickest way we can get started with osquery?*

- Common methods of interaction, workflows: via logging, ad-hoc via [osqueryj](#), via [the python module](#)
- For the optimal experience, [autopkg](#) install [osquery](#), sudo [osqueryctl](#) start, and collect logs in `/var/log/osquery`

## Brief history, similar tools, ICYMI

---

*'Prior art' presentations by the developers for reference, contrast with Puppet's [facter](#) and [osxcollector](#)*

- [Mike Arpaia](#) and [Teddy Reed](#)'s previous talks, teaser for my talk at [macdevops::YVR](#)
- ~~facter~~ is a purpose-built inventory tool to feed Reductive Labs Puppetlabs Puppet's... puppet ๓\_๓, akin to Chef's [ohai](#)
- osxcollector is associated more with [DFIR](#), and inherits from [osxauditor](#)
- Finally, what osquery actually **is**: 'objective C++' mixed with sqlite3

## OMG TMI BBQ

---

*After you've washed and moved all the large dishes out of the way, what's left are the teaspoons*

- Demo - why system\_profiler sucks, in specific versus auditing kernel extensions
- Discussing software like [santa](#), [SUIDGuard](#), and [rootfool](#), and event stream tables on [osquery.io/docs](#)
- More tooling around on [osquery.io/docs/packs](#), for pre-canned queries
- Even more demo'ing of python-wrapped ways of [grabbing browser extensions](#) and [an audit project](#)

## Demo Materials, Wrapup, Q&A, and Thanks!

---

Demo 1 - <http://url.aru-b.com/osqueryBrowse>

Demo 2 - <http://url.aru-b.com/adhocOsq>

*And the conversation's over once anyone mentions [Adobe](#), a.k.a. the Godwin's law of MacAdminery*