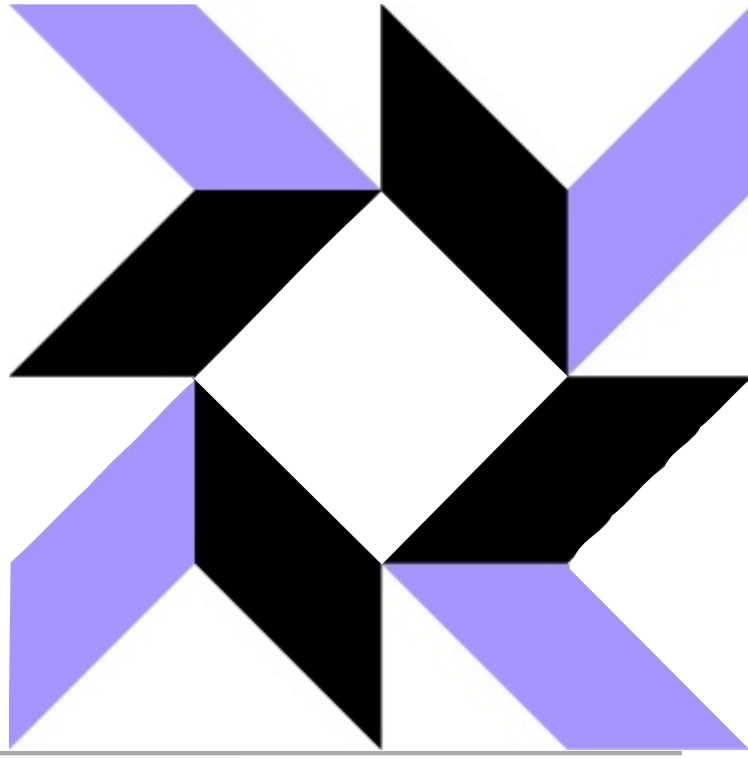
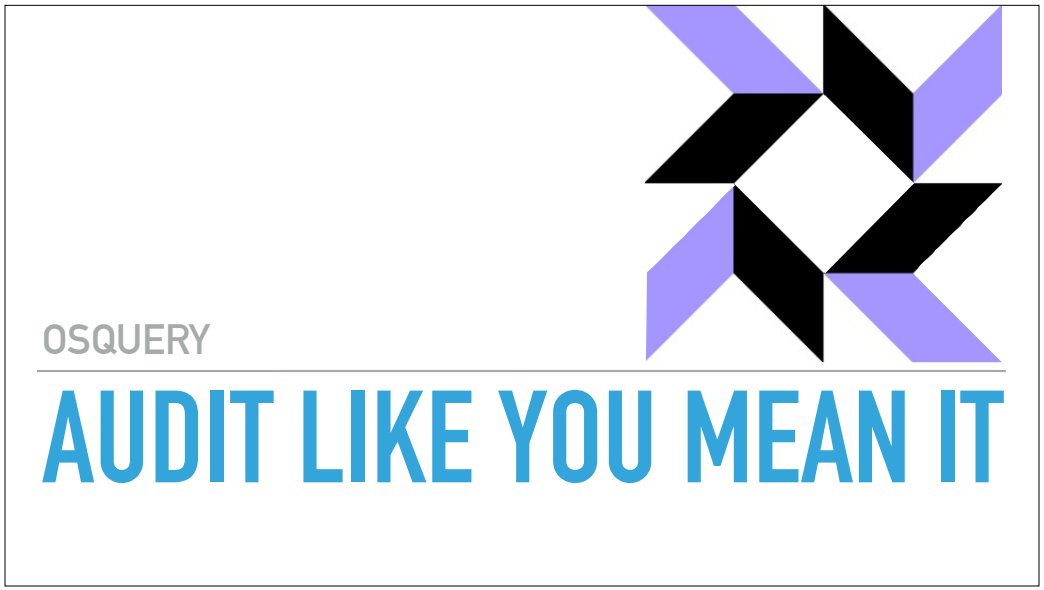


OSQUERY

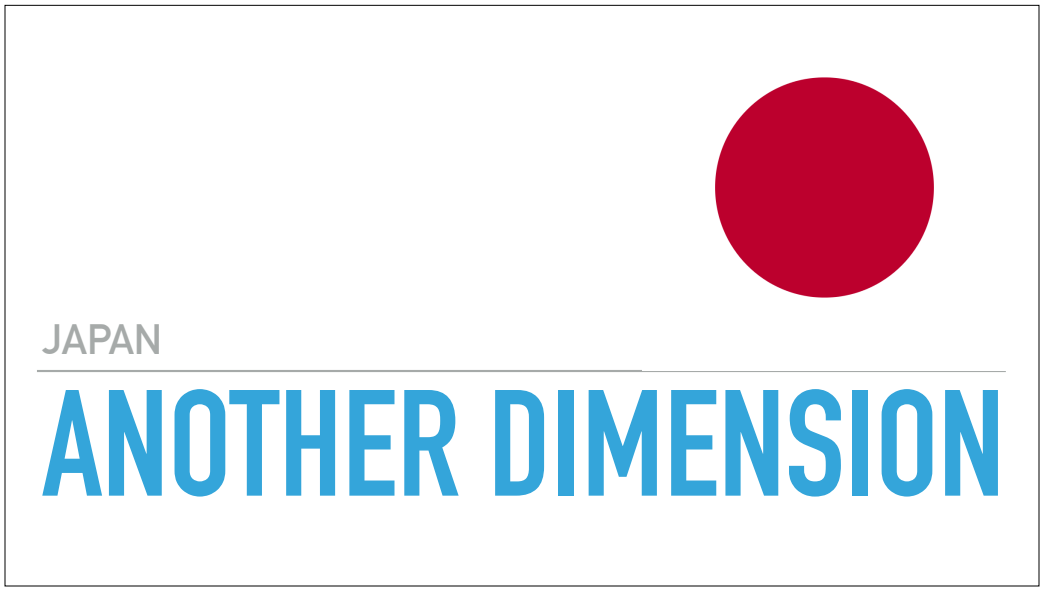
OR IT DIDN'T HAPPEN

Y'all may know me from such podcasts as the Frogor45, gotta plug the book with Charles Edge that I was able to do for Packt Publishing on iOS security, and here's more contact info. Regardless of that I'm not really an author, or radio personality, or performer, but I'd like to think I can play my position:

after getting that imposter syndrome out of the way, it seems recently that I am to osquery as Rich Trouton is to FileVault2, this is the third presentation I've given on it in as many months, but hopefully there's enough new material to make this one special. However... I've been somewhat regularly involved in the autopkg project and gave a presentation on it for MacTech2014, where I earned such rave reviews as:



Thanks to Professor Xavier not just for having me, but especially because he didn't freak out when I said I'm replacing the coffee in this session with Folgers Crystals - I actual want to slip in a completely different presentation than advertised, between the osquery stuff, and that's the one that I've been giving informally in conversation for years, the thesis of which is…… Japan:

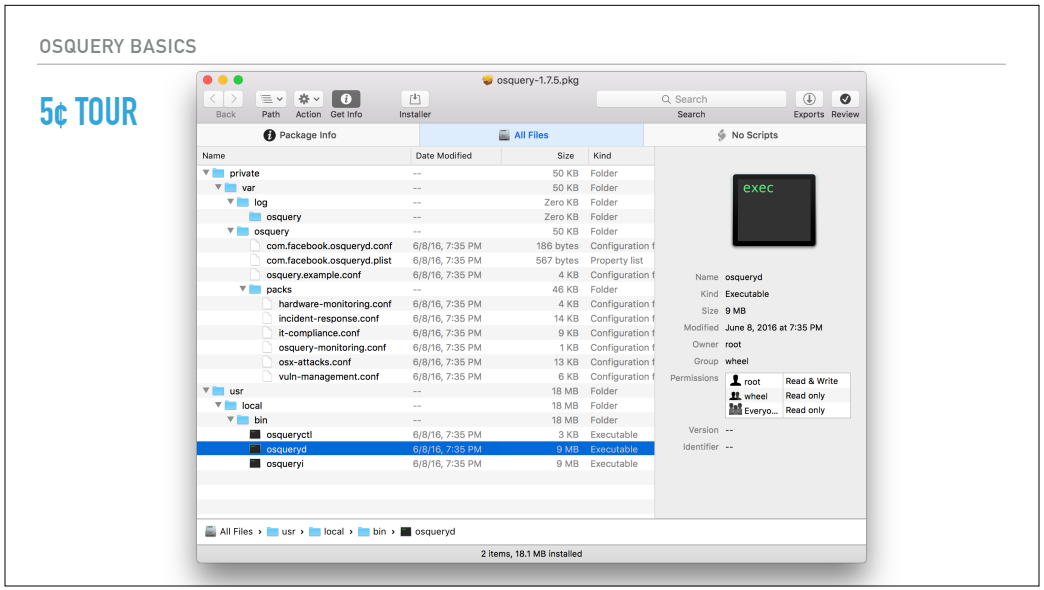


Another dimension. Because they have a lot of the same stuff we have here, they're just doing it differently. I'm a bit of a nipponophile, as my wife's family is in Japan and I've been going there yearly for 15+ years. I'll be talking about the Japanese language in particular, I expect everyone to pay attention because this will require audience participation, but before I get completely off course, there are references I won't be going over from the osquery side of things, so I should just point to them:

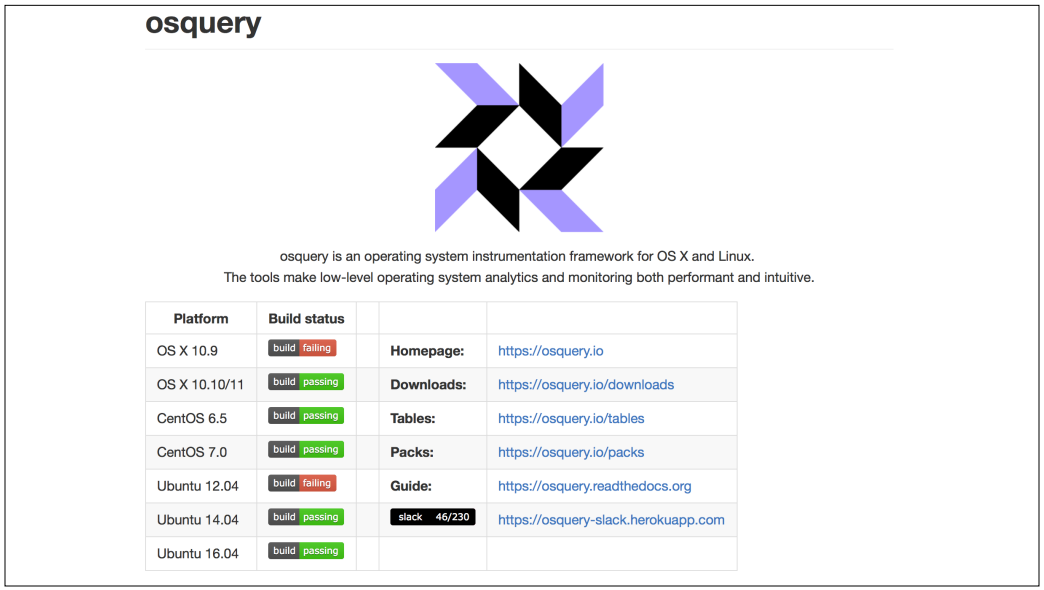
My talk at philly in April went over integration points between osquery and python, so see the handout for links to that video and the slides, last Month I went over selling osquery to your security department for the UofUtah which was mostly covering the security-focused whitepaper I wrote, which I've now committed to an organization on Github called aoswhitepapers for short. I got my work to evaluate not just osquery but munki, Sal, and crypt, I've delegated MunkiReportPHP and will probably be pushing Santa up there before the end of the year. Links to all of those are on the worksheet I linked to.
Slipping into the japanese…

‘oke’ in japanese is a borrowed from the western Orchestra, and then shortened because, just like three-letter acronyms, it's hip to shorten things. So the word you see right there, karaoke, is this bastardized amalgamation of a sound and a western word, and is written not in the character set borrowed from the Chinese called kanji, nor the phonetic alphabet for more traditionally japanese words - Japanese has one phonetic alphabet just for words considered foreign enough - it’s kind like how aliens in bad SciFi all have two eyes, ears, legs, arms, a mouth a nose, the way japan does language sometimes seems like an alternate reality. Kara is a description of emptiness

And then - it’s about to get a little recursive, so bear with me: Karakara can also be used when you’re thirsty to imply that your throat is empty. That repeating of a sound makes it similar to how Japanese does Onomatopoeia (which I just learned is spelled with a t in it?). and onomatopoeia means roughly a word that sounds like what it is, like meow or the tick-tock of a clock, nyancat is because nyan is the sound cats make. Japanese is full of these - when it's a heavy rain they call it zaa zaa, versus a drizzle, which makes the potspots sound. Muddy or filthy is dorodoro, and here on the bottom right is not really onnamonapeia, but sounds like it - it means small part, like earlobe, and I just love the pronunciation - bubun - so fun.

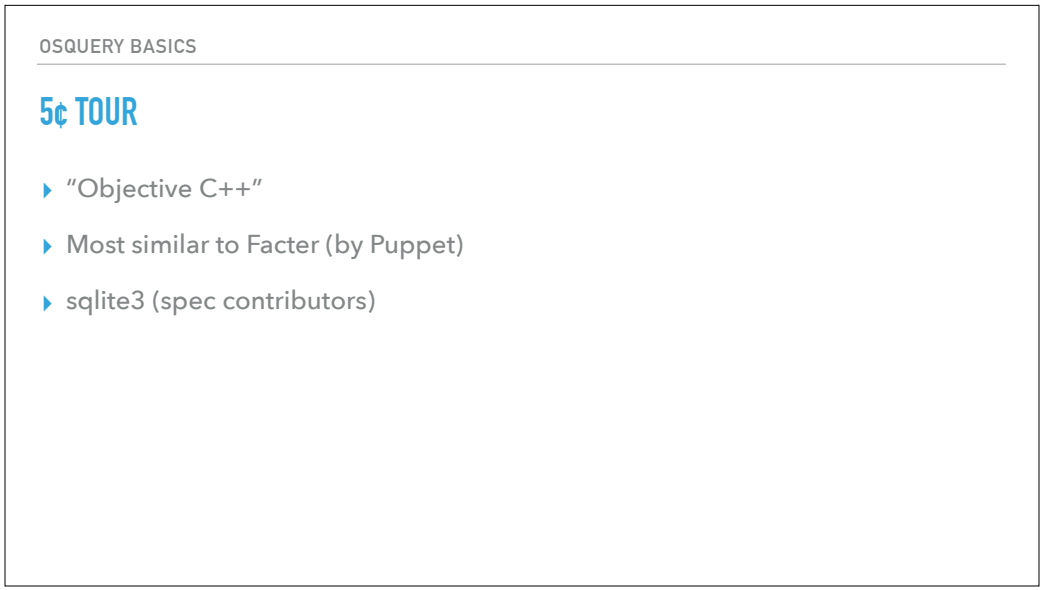


Ok, so here's the nickel tour or whirlwind intro to osquery. To cut to the chase, as you can see if you open up the installer in suspiciousPkg, primarily the tool is a pair of client side compiled binaries that you install to usr/local/bin along with a control script and a few supporting files and setup directories. You'll notice it does not drop anything into the root Library or LaunchDaemons, you can use most of its functionality without sudo or giving it persistentance, and you notice the signed pkg they distribute has no install scripts.



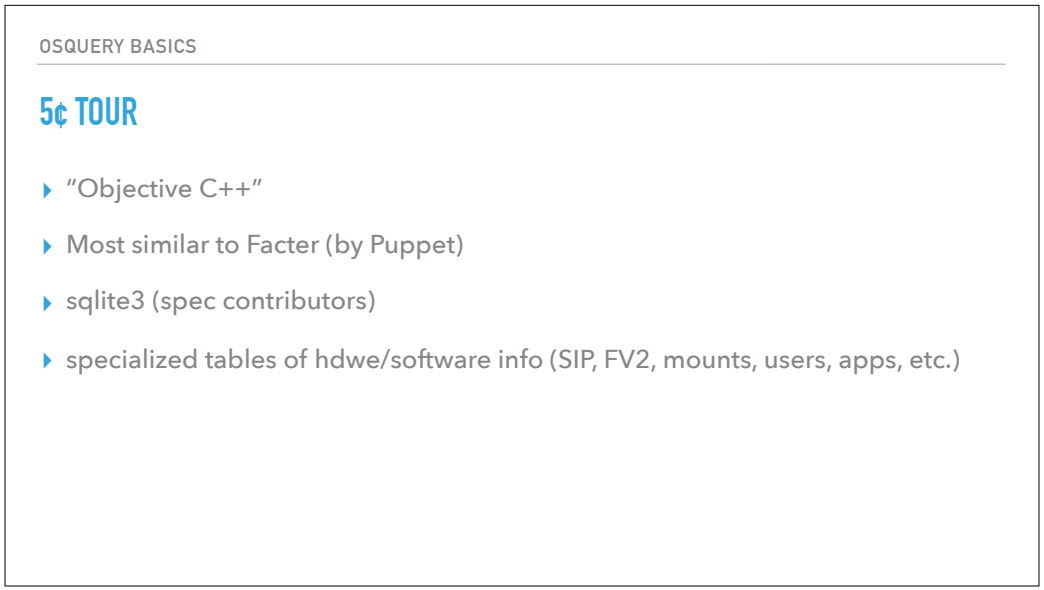
Facebook is actively supporting the project, and allowing its main development to be in the open on Github. They have a relatively quiet Slack instance that their C-level security czar sometimes hangs out in, as if that's some kind of endorsement. There’s also the entire osquery.io website for docs and a cleaner presentation of the project.

It is written in, as the joke goes, ObjectiveC++, As security of the macOS was a priority when starting the project so it therefore holds a central focus. It officially supports Debian and Enterprise Linux variants as well as macOS, with Windows support very close to being official, among other distros. It has been the subject of external audits, a highlight of which for me was there were links to specific files at specific commits on Git being referenced in that report which they publicly released of the audit - that feels very modern and open and correct to me

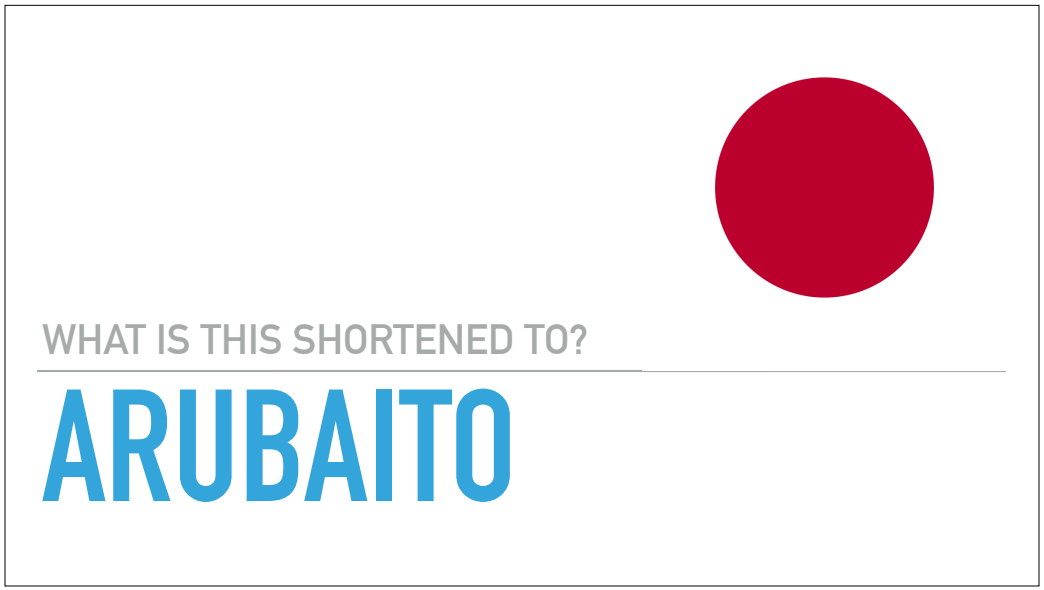


and not to give it the oversimplistic comparison like when people call a new startup 'Uber for Netipots'...the closest tool conceptually out there is Puppet's Facter, which is also written in C++, but while it's used for information gathering, it's more often a way to make decisions to act on, which I'll come back to in a few bullets…

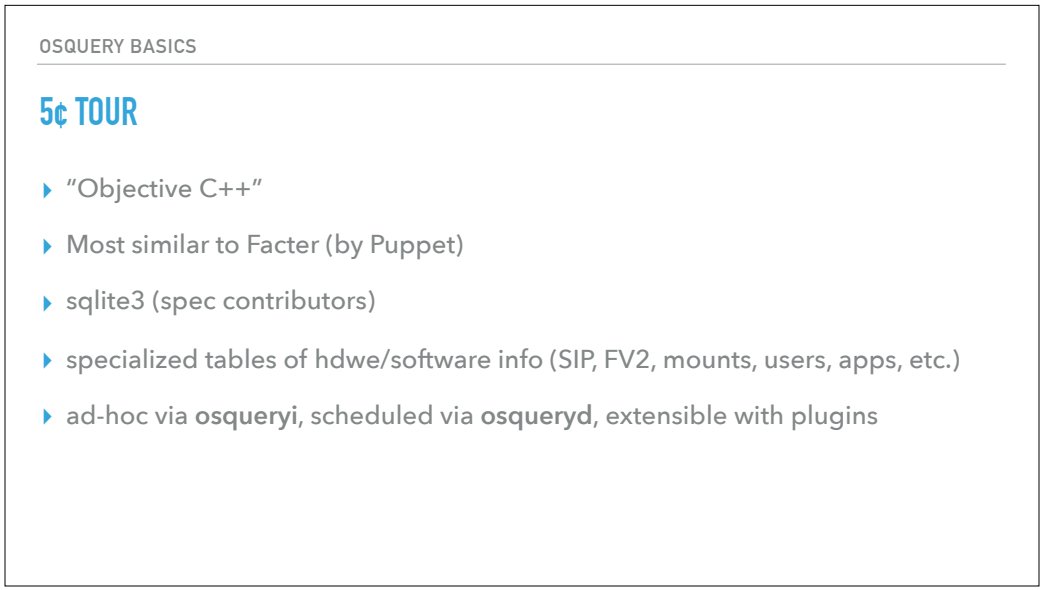
What makes the design of osquery kindof crazy is that it groups classes of things you'd commonly report on as sql tables, and actually relies on sqlite3 to the point that they contribute to the open source spec so that they can push performance and features of sql forward.



So when you want to interact with osquery you're actually writing sql that does queries on specially organized and instrumented tables full of security-specific auditing data. firewall SIP, launchd jobs, kernel extensions, hostfile contents, etc, and some of those you'll do rich filtered or focused lookups by using the SQL concept of 'joining'- for example you can join the filevault encryption table against the 'mounts' table if you only care about the boot volume, or for apps you'd find in the users libraries app support folder, you can join against the users table to enumerate turrible stuff like zoom.us



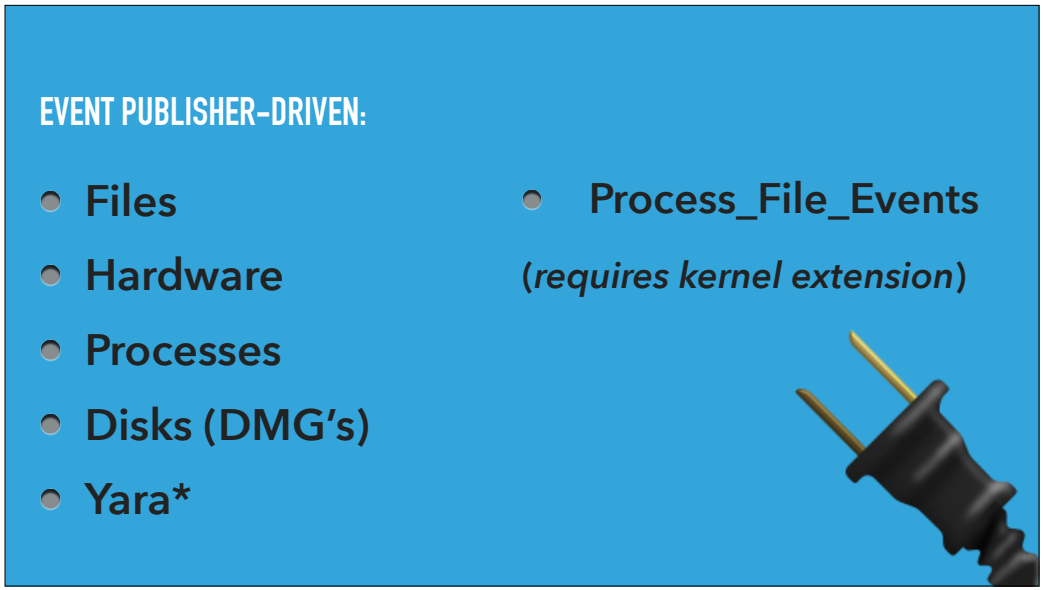
Quick intermission, two bullets to go, but your next chance for audience participation. this is how you refer to a part-time gig in japan. How do you think they shorten it?
Some people here may know where the Japanese actually borrowed this word from, and it's Henry and Andreas - I think they pronounce it Arhubeit. So it's 'baito' this is more to say the Japanese will borrow from lots of different western cultures, not just english.



Okay, back on track, I told you it's sql statements that make up how you interact with osquery, but where do they get piped through, how you actually get results is one of three things: first interactively, meant for while you're iterating over what's the prefect query unit of info you want to collect, and that's with osqueryi, an interactive shell, or second, you can leverage the daemon, usually along with scheduling via a launchd job, which gives you slightly more features, and finally not only can you use the python module to generate tables of information, you can open up an instance of the osquery daemon from python to shove all your queries down all at once, rather than shelling out to it or queueing up a whole bunch of process opens individually. A rule of thumb for sql is feed it as much as you want, it's usually arranged internally so you're not going to overload it, its handling of queries will be optimized for as much RAM and CPU as you want to throw at it. And the osquery team uses profiling tools to ensure that they are super low-priority on the system and cause minimal system impact.

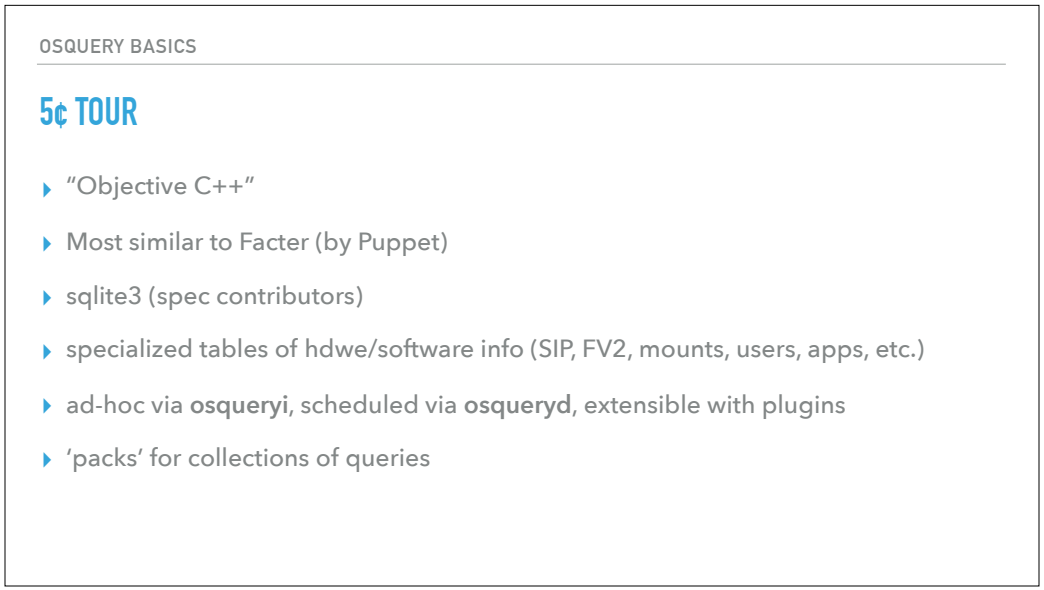
The title of this talk is ‘osquery or it didn’t happen’ because there’s the common dilemma of how does a scheduled check catch stuff that happened in between the interval that the queries actually runs on - if something malicious happens but puts everything back where it found it, how would you know? Additionally how does any kind of agent slurp off activity data in a low-resource enough way that it doesn’t noticeably impact the system the rest of the time, especially when the customers on the machine are developers that check every kilobyte of network traffic?

This icon is for one of those tables I mentioned can take a queued or cached stream of data written to a local instance of another Facebook product called rocksdb then on the scheduled interval write the logged information out. Your tip-off is the name of the table will say 'Events' in it, and therefore you have this subscription to an event publisher type relationship to make sure you don't miss occurrences of whatever you want to track.

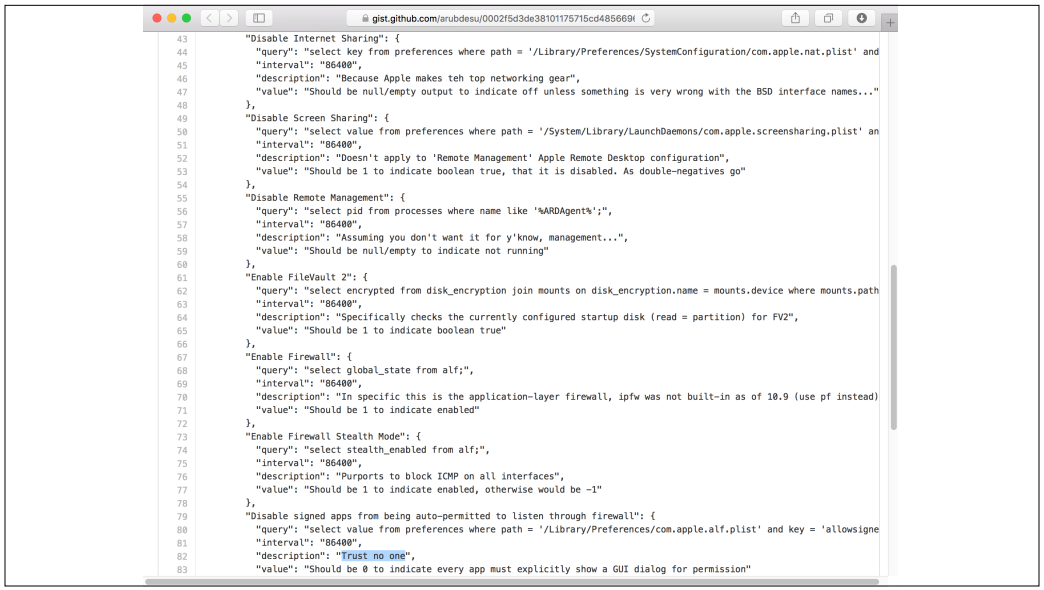


And here are the ones you get by default, the DMG one is Mac-specific, and there is a kernel extension they use internally at facebook on linux and mac, which that process_file_events table relies on. That kext is represented in their docs by a plug icon.

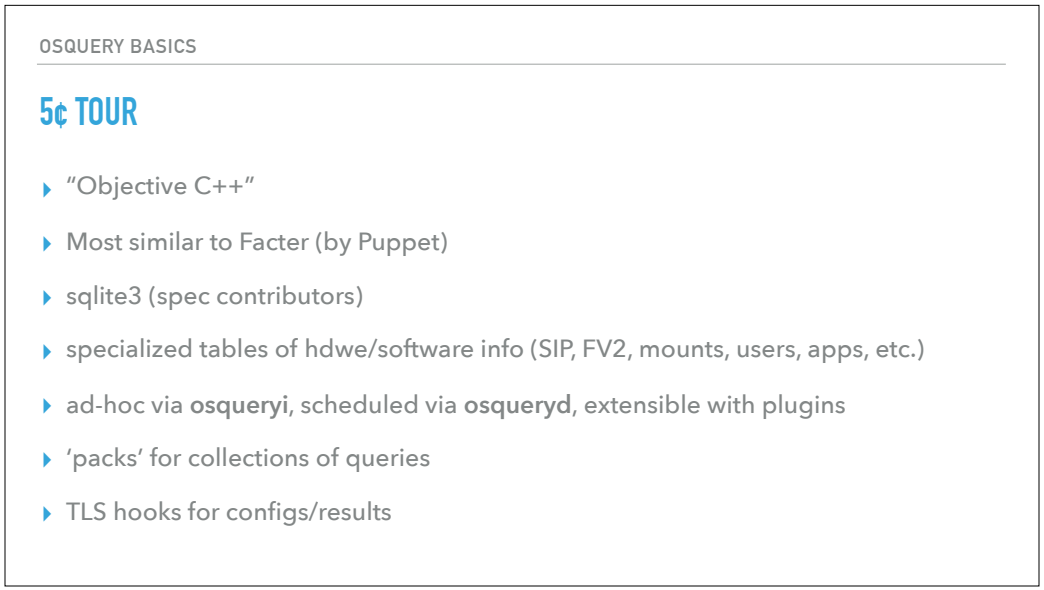
Breaking news, the yara events table is interesting because if you're running a redacted version of the operating system you may just find that xprotects signatures could be based on yara now, and just possibly at the path you would find there, I dunno. Pre-release software is not exactly something to bank on, but interesting stuff that osquery already had taken into account since early on



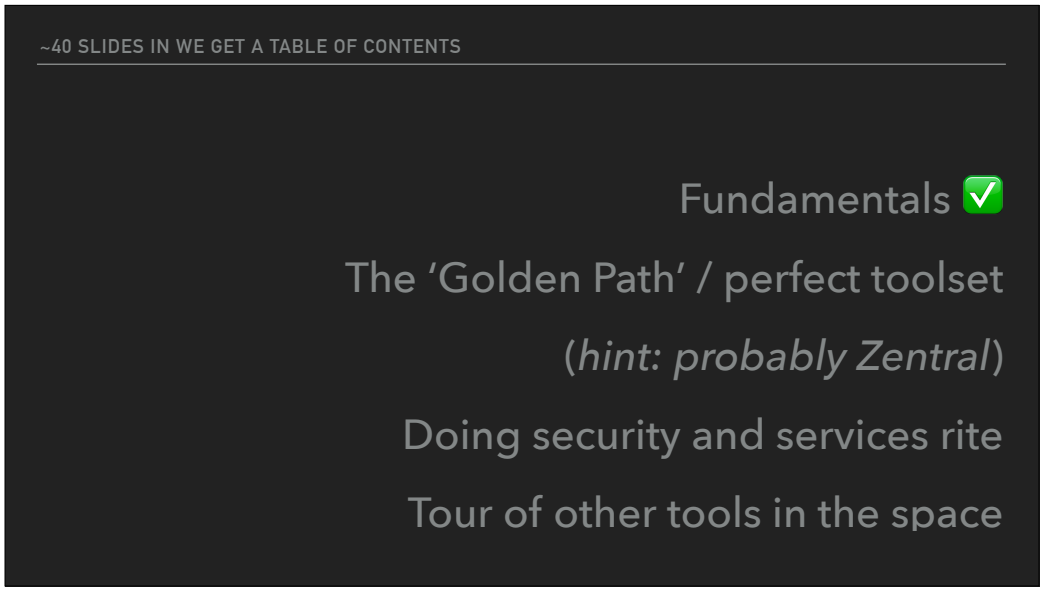
Back to the nickel tour, one last point is that you can arrange your queries into units called packs for ease of distributing to a bunch of hosts as a unit of configuration, and inside the pack there's even this advanced concept of a query that checks a condition before running the rest of the checks, so you can distribute this to you linux systems and Macs and not worry about the stuff that's only applicable to one platform. So that feeds into my earlier point about the similarity to Facter, it can do its own little preflight to tell if a query is applicable to be run on the system in question.



And here's an example of what a pack looks like, I made this one for the osxlockdown project if you're familiar with that. The link to this is on the handout as well.



And continuing the distribution theme, osquery can use a lighter-weight set of flags when launched locally but check in with a secured http endpoint over TLS, just like Zentral, which you'll be hearing about after me. Not only can it push configs out, it can get results back in a secure and well trodden way of communicating over the WAN, aka with web-specific technologies.

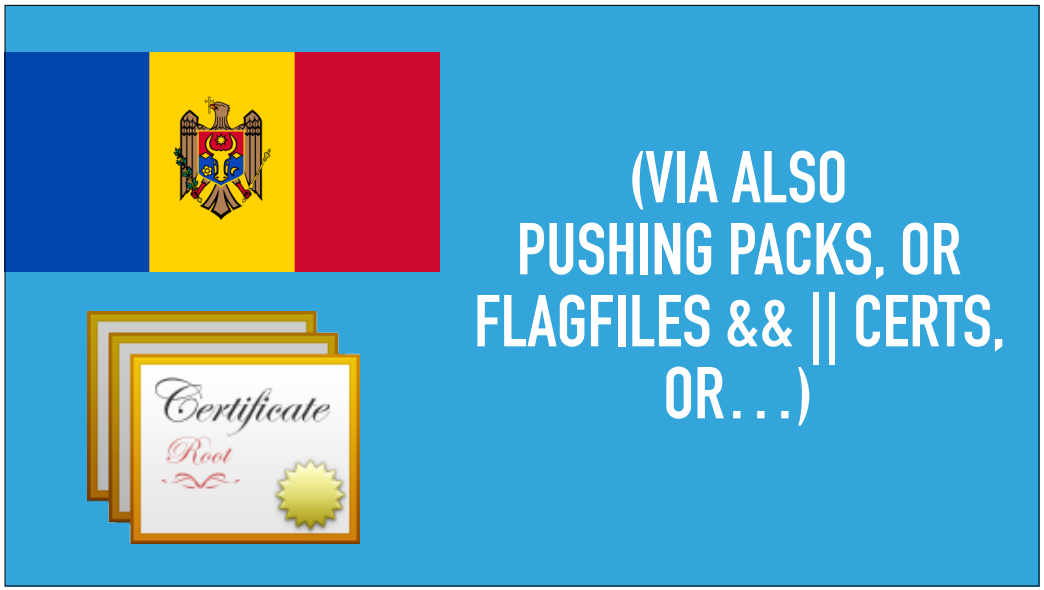


As we mostly covered foundational, I'm going into some conceptual quote-unquote 'rite' way to implement services that helps us scale out osquery,

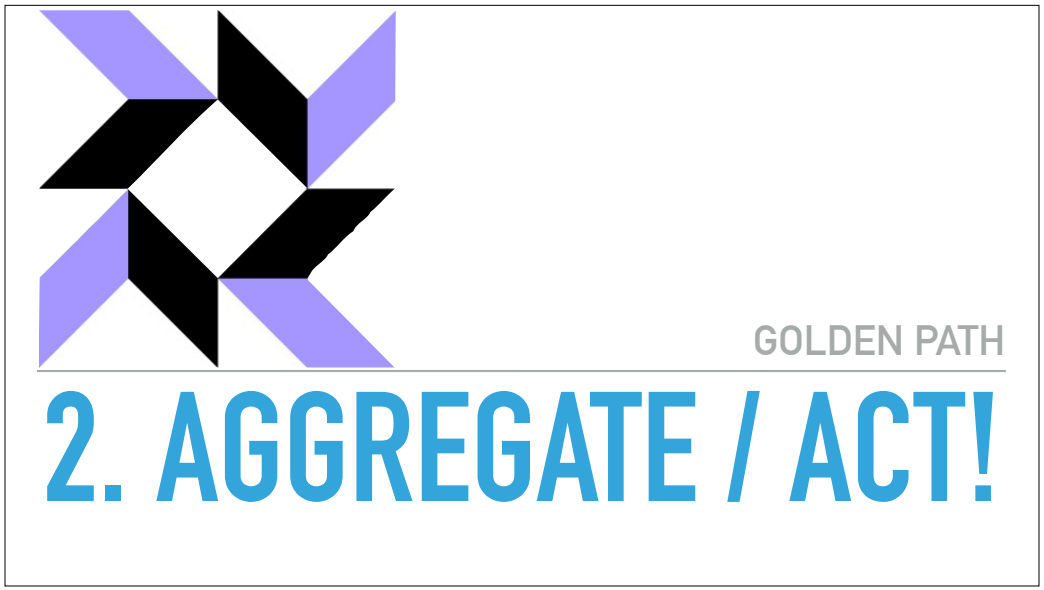
transition to some more hand-waving theoretical stuff about security in general,

and then finishing with covering the current tools in the osquery mgmt space I’ve run into in my travels. And most tools want there to be a quick way to get started on the quote-unquote golden path, so…

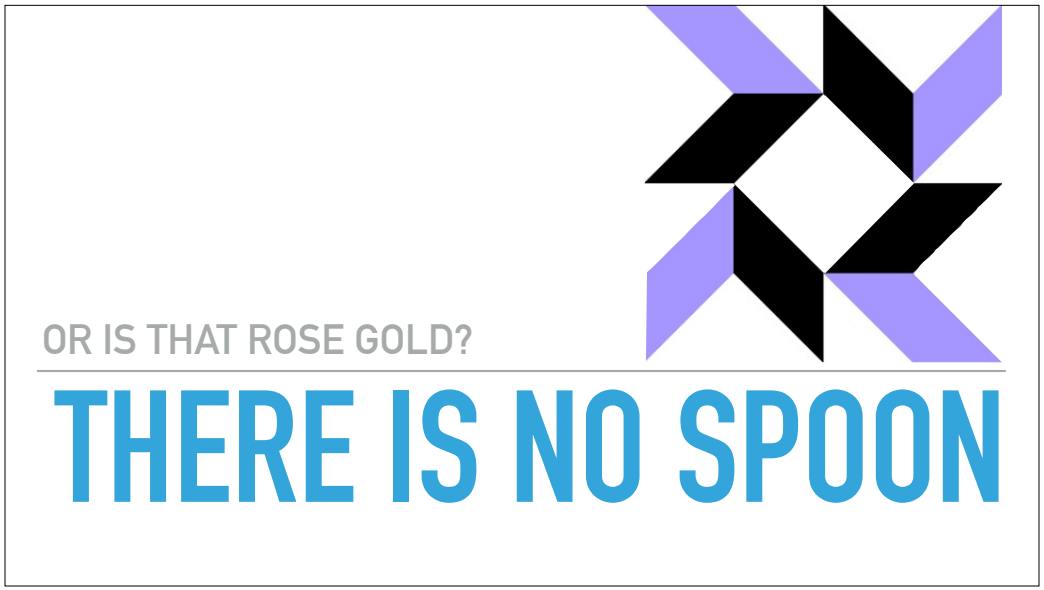
First! push your pkg, customizing the launchd jobs to rename them from 'Facebook' if you have privacy-paranoid, tin-foil hat wearing customers.



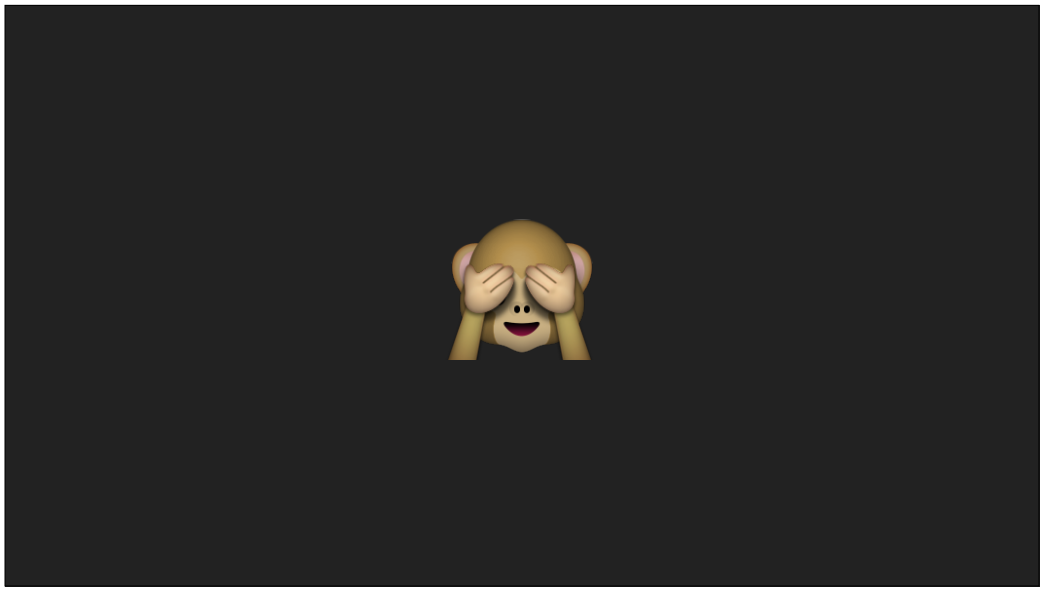
And you can push down that config with a whole bunch of queries or query packs as I mentioned, ideally maintained by config mgmt, or just use a flagfile that you'd pass to osqueryd (plus certs). that means you just set some options to auto-load from the launchd job that helps it check in, get configs, and send results, as I mentioned earlier.



Then! figure out what your anomalies are or the compliance baseline is and how you want to act on that information. Which is kindof like press button receieve bacon, but if it sounds oversimplified as a golden path,



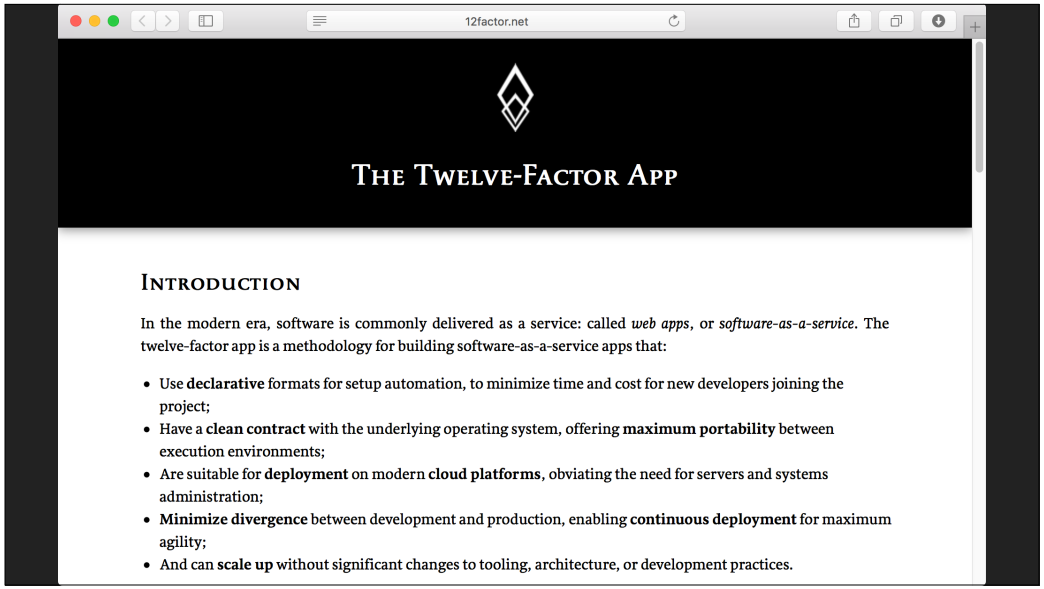
It’s probably because it is, the challenge when choosing what infra to stand up for osquery is that some customers want everything: security, privacy, and flexibility. One deploy method may not fit all, it may be that standing up yet another agent or ingest infrastructure isn't attractive, or straightforward if you don't have the rest of your requirements and homework done from a threat modelling perspective.



To expand on that point of not just being able to push the pkg as-is, osquery considers shell history an example of something that an average user wouldn't consider private, but you can remove that table if you want. Browser history is an example of something the core osquery devs consider an anti-pattern for tables of things they don’t want in the main distribution. Maybe process envs are another thing you want to leave out, depends on your environment, so devs and sysadmins don't leak credentials or other config data to your aggregation system.

sysadmins may even be concerned with WAN-facing log aggregation being exploitable, or cert infra being yet another added mgmt overhead to deal with

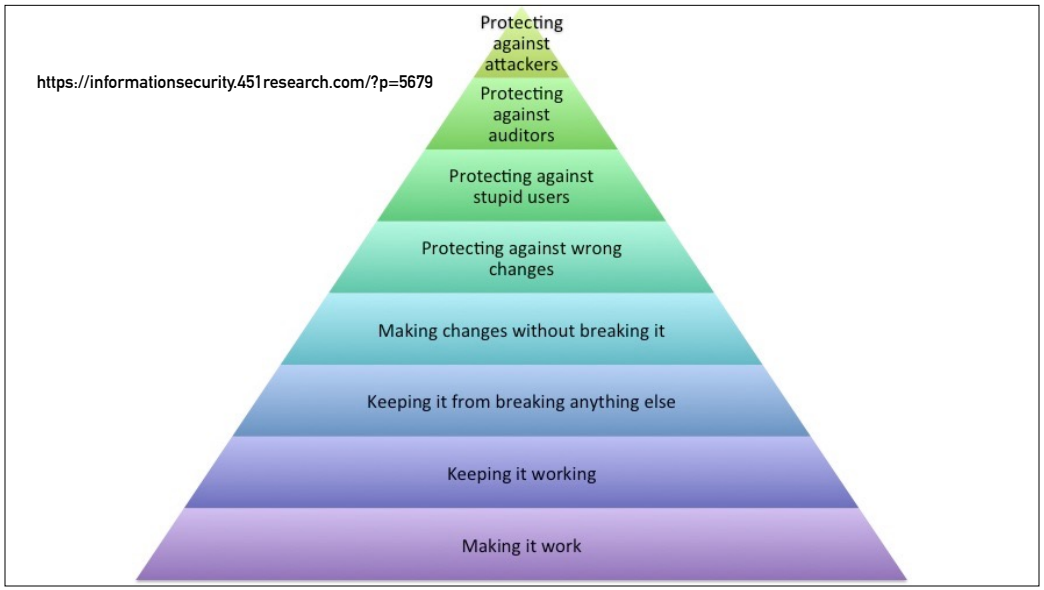
And so I wanted to detour into security via the service of a managed osquery deployment. I want to be practical about finding a deployment model that fits - I mean, this is MacDevops, so it may be a safe assumption we all know about standing up customer-facing services...



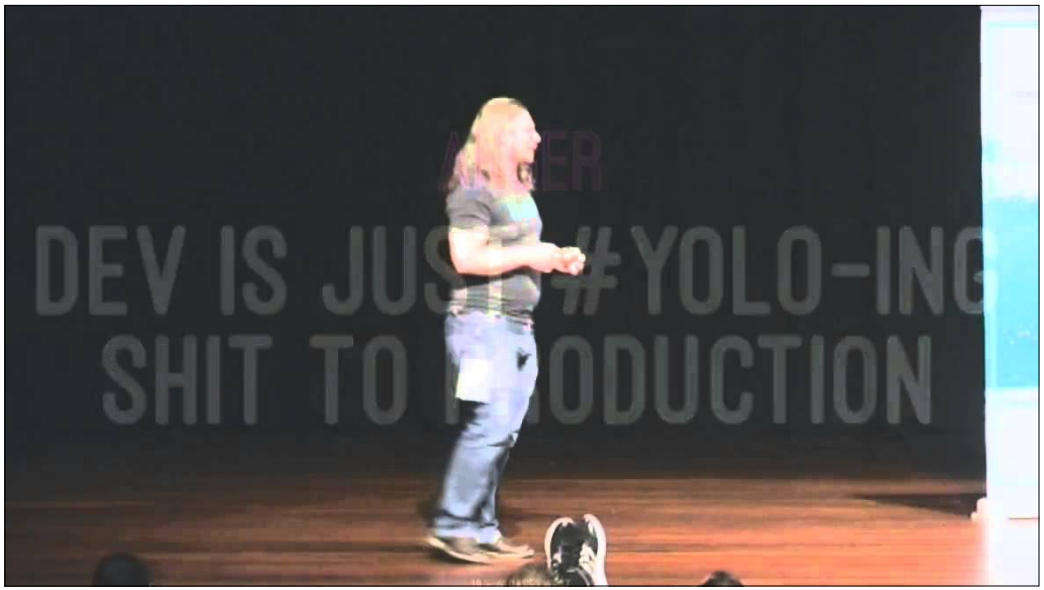
And this is where I just drop buzzword new hotness stuff, but I'm talking about a not-insignificant hurdle folks may have if they know what robust services require. If you have two data centers and not just a network closet, you may already know how to deploy customized services. There's the kernel of what we need in this concept of The 12-factor app, which sets proscriptive guidelines for standing up a service. And if you're hosting it yourself you have to accept you're essentially trying to do as good a job as a platform-as-a-service vendor like Joyent, Digital Ocean, Pivotal, AppEngine, or the 900-pound gorilla AWS, which we get to hear a bit about this year.

Talking about this heady, conceptual stuff is only important in the way it helps us understand why things are built they way they are, and just as much as osquery seems like the alternative to shitty antivirus - if you're lucky you missed this disclosure about every version of symantecs scanning engine could bluescreen a machine

on linux servers theres the OSSEC framework that, while respected and widely implemented, from what I hear it is not nearly as nice to interact with. Sysdig's Falco is another product that lives in a similar space that Henry hip'd me to.
But osquery's also not packed with similar features to GRR Rapid Response, which allows you to quote-un-quote 'hunt' machines for indications of compromise over the live, latent, crappy WAN. Which poses the question do we need these feature in osquery?

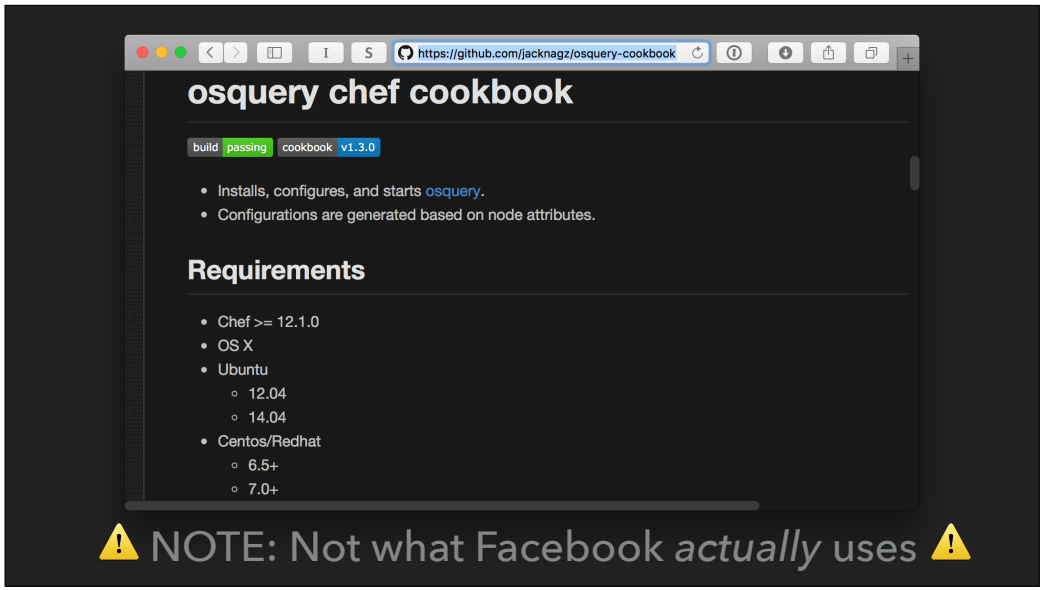


To bring it back down to earth for what the core competencies of IT are in relation to services in general and security in specific, this is from a post by Wendy Nather.
reads slide



I therefore don't expect everyone to prioritize osquery or the systems to support it, there may be lower-hanging fruit to deal with first. But, if you're in a situation where you have the luxury of being able to approach stuff like log aggregation and periodic privileged audits on the workstations under your management, make sure all the talking across the network is secured, that there's trust and the ability to revoke it or have it be short-lived enough that you can prevent misuse. You can also drop out the things from osquery you don't want to see so privacy can be respected where that is a concern and liability can also be contained.

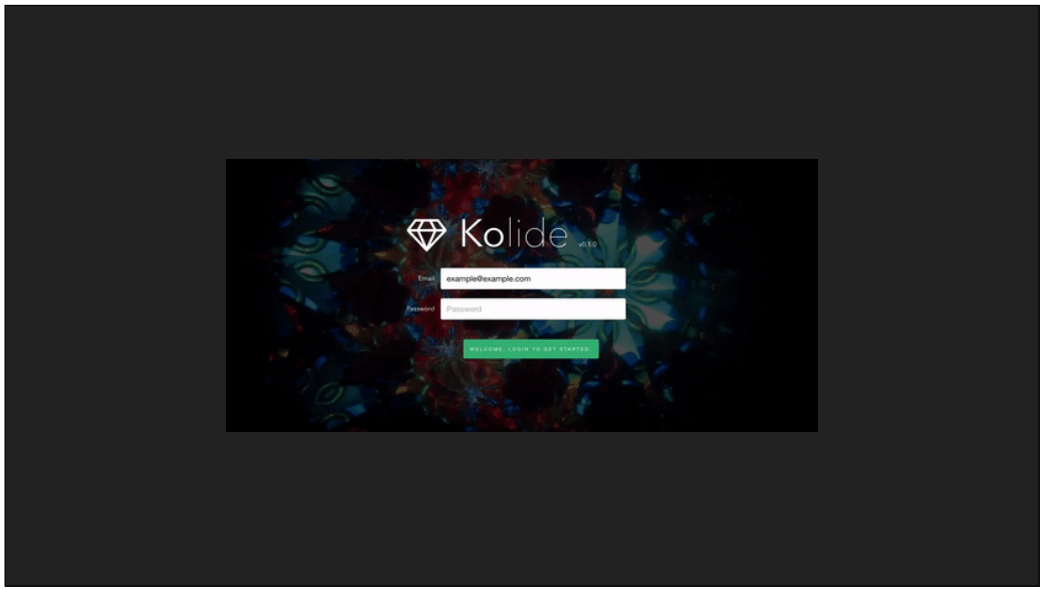
Alright, home stretch, I'm just going to fly through a few mgmt systems, starting with an approximation of what facebook is doing:



As we at MacDevops know, they use chef for many things and platforms at Facebook, so you could imagine they distribute packs and the configs with something like this cookbook which is not actually what they use,

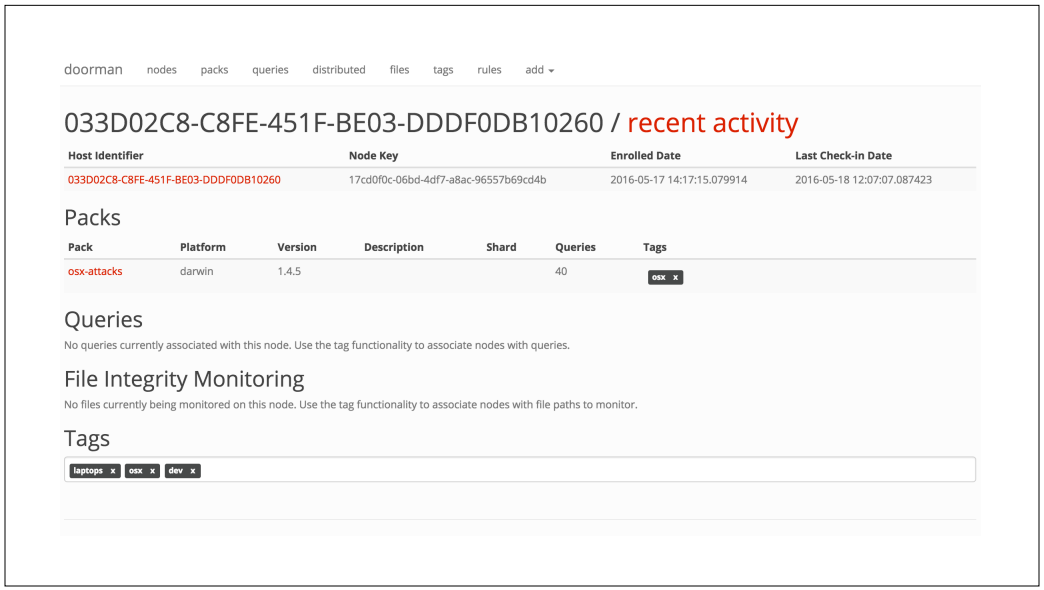
And they also don't actually send the logs to Splunk, but they have a similar tool to provide aggregation, filtering, and reporting that can then feed whatever monitoring, helpdesk ticketing, or infosec remediation process they might have. Which sounds like a pretty complete set of tasks, if you're already leveraging logging infrastructure, and/or don't mind dealing with the weight and agent required to ship the logs you're generating, and you trust your config mgmt tool to put all of the queries in place for you. Which is reasonable, but...

Besides Zentral coming up next, there are two other tools I'm going to mention, one still very beta with a polished UI built in Go, which I'll call the flashy way, and the other using certs and TLS and a scalable python flask architecture I'll call the volvo.

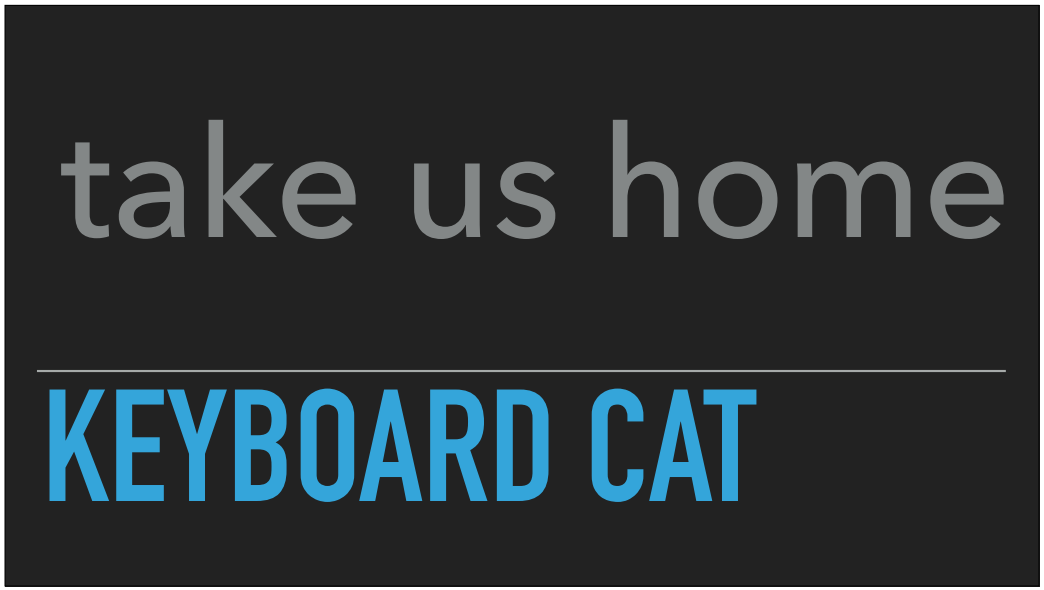


First up is kolide, which is a do-over type evolution to the product envdb by the same developer. It has a few assumptions based on what its intended use MAY be: live connectivity to nodes, as if you're running phpMySql on them, live over the interwebs... i'll get back to that. It uses a makefile to call docker up for the postgres and redis dependencies which makes it super simple to get started with. Caveats: it's still very rough, one would assume it's for a GRR-style use case, but it's so cool to imagine the concept of instrumenting a flock of machines over the live internet. It doesn't solve the message bus and reachability issues, which mgmt tools like chef and puppet help enforce for you, but, different use case

I've not demo'd windmill but I checked with Scott Roberts from GitHub and it's another TLS-based distribution point, but there's a PR supposedly being merged to allow you to use the TLS endpoint to forward results to Prometheus for trending or ELK or spunk or Greylog or whatever aggregation and reporting console you have



And then there's Doorman, which is a bootstrap-wrapped mgmt console to both apply queries to uuid-tagged nodes or perform adhoc one-off runs called distributed queries. The lack of identifiability or inventory you see in this screenshot is probably a design + according to the devs, because this is the super-interesting (to me) constraints it has: BYOD machines, no other agents installed, no logging by default, send the results via that same TLS connection. Yet the auth story is really good and flexible, secure administration and stability were definitely prioritized in its design. Concepts of groups is replaced by ‘everything is tags’, which is supposed to make more sense to me, but I somehow can't wrap my brain around. It’s conceivably used at Square and actually security product development houses that leverage patch mgmt and other non-BYOD use cases, very interesting although not so easy on the eyes, gotta say.



So, that’s your tour of tools the space and concerns or requirements that have influenced the design of other projects, and now it’s time for our wrapup

And it wouldn't be right not to mention I don't have these more robust ways to manage osquery at scale pushed out to production yet - the python based, ad-hoc methods are still the ones I mentioned at my PhillyMacAdmins preso, but I have puppet up - all I'd need is a logging stack to do as Facebook and a few other companies-that-make-the-internets have done. But development of these mgmt tools and osquery itself is a journey that's 1% complete, to use another facebook-ism. With the exception of Zentral there may not be a common-case service to manage osquery that's right for you - I hope folks will participate more, ask questions on whichever Slack you'd like, and push to let developers know what features are missing or important to you, what could be better from a docs perspective, or pitch in if you're savvy with contributing to open source, as Joe Chilcote will detail tomorrow. And since I actually have time left for QA, let me blow that away with two last Japanese factoids:

Thanks!