osquery for Mac Security Whitepaper

Prepared by Allister Banks. Last updated May, 2016

Background

Facebook started the osquery project after assessing the products available to secure their production Linux servers and Mac fleet. Nothing on the market allowed them to have adequate performance and deep inspection via native APIs, nor could they discover the state of systems while subscribing to events at scale (hundreds of thousands of systems). This whitepaper details its use specifically on Macs.

Executive Summary

The osquery project allows inspection of Macs by enabling administrators to run sql-like queries on a running operating system. This allows anomaly detection to happen in real time, as logged by its daemon, without sacrificing performance. To share their methods and collaborate on the code base, the Facebook developers who maintain and continuously extend the capabilities of the project develop it in the open on GitHub[1]. Code review is performed on every check in, and a consistent release process ensure rapid improvements do not affect stability. 'Packs' allow community members to collaborate on common things to detect and report on, which facilitates timely flagging of 'indications of compromise'. This relieves the engineering and administration burden of the community as a whole, and makes securing systems as per industry standards like PCI-DSS and HIPAA a maintainable process.

Explanation of Features and Concepts

While the product can also be used on Linux and Windows systems, we will only be covering its use on Apple Mac workstations. The osquery toolset contains different moving parts that are employed based on how it will be used, and are conceptually arranged around its database-like model. Perhaps more optimal than tracking down changes during incident response, osquery can report on the state of an operating system over time. It does not attempt to block functionality or remediate when a OS is considered out of compliance. The most fundamental concept of its model to understand are the tables.

Tables

Just like a regular database, data returned by osquery is organized into tables with rows and columns. The criteria for some tables are separately geared toward each OS, and Darwin (the kernel underneath Mac OS X) has the majority of ones that ship with osquery. Platform-independent criteria can be of use to ensure that e.g. DNS for a specific host has not been hijacked, among many other concerns like running processes. Mac-specific tables are added over time, and the ones currently shipping with the release version are listed here: https://osquery.io/docs/tables/#darwin

osqueryi

To interactively query the tables once the tools have been installed, you can run the osqueryi command in a shell like the Terminal application on a Mac. You can ask for things like what tables of data can currently be returned on the computer by using the sqlite-style syntax for commands.

```
$ osqueryi "SELECT cpu_brand, physical_memory FROM system_info;"
+----------------------------------------+------------------+
| cpu_brand                              | physical_memory  |
+----------------------------------------+------------------+
| Intel(R) Core(TM) M-5Y71 CPU @ 1.20GHz | 8589934592       |
+----------------------------------------+------------------+
```

[1] https://github.com/facebook/osquery

osqueryd

By leveraging a background daemon that will either keep your queries running on a schedule or keep tabs on an event stream (for the criteria that requires constant monitoring), osqueryd is the recommended model to use when deploying at scale. Paired with logging and shipping those logs to an aggregation endpoint, it becomes a way to get real-time metrics back in the same way critical server systems would be monitored.

Plugins

Using C++ allows osquery to interact with the native API of the operating systems it supports. Getting data in to osquery via tables is important, and lowering the barrier to entry has been a priority as well. In addition to C++, developers looking to contribute can use the thrift API to build tables and interact with osquery, which is bundled with osquery-python[2].

Deployment and Vulnerability Detection

The plugin framework referred to above can also be used to get logging data out of osquery, but to approach this topic we should talk about the breadth of what you can do with the tool. Even scheduled checks of the presence of issue can only be so instructive. Identifying trends becomes important as possibly benign behavior can become compromised or turn malicious over time. osquery therefore has a daemon you can configure to accept a stream of events.

A common workflow is to take the desired output discovered during an osqueryi session and configure osqueryd to subscribe to the state of that criteria. This is more commonly paired with a log-shipper so that the stream of events can be processed off of the host, and the same plugin framework described above can allow integration with different systems like Splunk or elastic.co's Elasticsearch.

Packs

Currently the osquery project maintains common interesting criteria to check for when scanning or subscribing to events on Macs, grouped into a file format called Packs. They are leveraged by osqueryd, and the first ones released by Facebook have headings like 'incident-response', 'it-compliance', 'osx-attacks', and 'vuln-management', found here: https://osquery.io/docs/packs/.

Facebook has used their tool to proactively secure their Mac fleet. Examples of using osquery for vulnerability detection include the 'wirelurker' exploit discovered last year on Macs, which Facebook themselves demonstrated a detection process on their Protect the Graph site[3]. Another example was an

---

[2] https://github.com/osquery/osquery-python

[3] https://www.facebook.com/notes/protect-the-graph/anomaly-detection-using-osquery/1532788613627951

exploit in June 2015 that abused keychain access control lists on the Mac, which was also shared on the same site[4].

Industry Specific Compliance

HIPAA

As specified in the Administrative Simplification 164.312(b)[5], the standard to implement software controls that record and examine activity on systems which may contain or use electronic protected health information plays into osquery's strengths. Many of the other recommended standards are already included in the tables that ship with the product, all of which can be employed with minimal performance impact.

PCI DSS

For known and not-yet-disclosed vulnerabilities like those exploited by adware, malware, trojans, and viruses, osquery can detect indications of compromise the moment they begin displaying that behavior on affected systems. This is in line with PCI DSS guideline 10.2.7, regarding logging the creation and deletion of system-level objects[6].

NIST 800-53

Working hand-in-hand with configuration management tools that ensure compliance like Puppet or Chef, osquery can also detect that the enforced firmware or configurations are not modified over time. Cryptographically proving the signature on objects is made possible through its file integrity monitoring, which uses the approved SHA-256 NIST algorithm, as dictated by security control SI-7(6)[7].

Conclusion

With its unique tailoring to the Mac platform, focus on performance, and responsiveness from a community of developers employed to maintain it, osquery is becoming an essential monitoring tool for large installations. It is the manifestation of the idea that no vendor can detect all possible vulnerabilities, and efficiently profiling the effects of a compromise helps you remediate with full historical accuracy.

---

[4] https://www.facebook.com/notes/protect-the-graph/detecting-unauthorized-cross-app-resource-access-on-os-x/1619875274919284

[5] http://www.hhs.gov/ocr/privacy/hipaa/administrative/combined/hipaa-simplification-201303.pdf pg. 67

[6] https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf pg. 86

[7] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf pg. 40