Chapter 3: Creating A Repeatable, Reusable, and Reliable Security Data Analysis Workflow and Toolkit

While the majority of the content of this book focuses on how to analyze and visualize “security” data, there is a rhinoceros in the room1 that must be dealt with. As we saw in the previous chapter, if you aspire to become a security data scientist, it is not enough to just take snippets of code, apply them to your own data samples, produce a graph (or two) and declare victory. Just as the comic book hero Captain Marvel2 took on the powers of six mythical figures to save the day, you will need to be infused with the skills and abilities of a librarian, systems administrator/architect, mechanic, programmer, project manager, and forensic pathologist to gain the most personal and organizational value from the concepts and techniques presented in these pages. Unfortunately, it will take a little more work than uttering “*Shazam!*”, but this chapter should help you down the path of acquiring those fundamentals skill with the added bonus of no lightning strikes.

Maintaining an Inventory of Data Sources

It would be difficult to say that no data exists for us to process for the purposes of security-oriented analysis. Even the most nascent security analyst should be able to rattle off a list that would look similar to Table 3-1. But, do *we* really *have* data? Network administrators are usually the owners and caretakers of their device configurations and logs. The same is also true for Windows and UNIX/Linux administrators. The security team may own firewalls, but if you’re a large organization, that may be a completely different team than the one that will perform analytics on the data.  Unless you own the data from generation to deletion, you will be relying on others to either provide it or provide access to it.

Table 3-1 Potential "Security" Data Sources

|  |
| --- |
| Windows Event Logs  Linux/UNIX syslogs  Mainframe Logs  Network Device Logs  Proxy Server Logs  Firewall Logs  Anti-malware Management Event Databases/Alerts  Vulnerability Management Databases  Patch Management Databases  System Configuration Logs  Identity & Access Management/RBAC Records  NetFlow Data  PCAP Data  HR Data Feeds  Application Logs  Web Application Firewall Logs  E-mail Gateway/Spam Filter Logs  Business Transactional Data Logs  Database Audit Logs  Asset Management Databases  Physical Security Event Logs  IDS/IDP Alerts  Indicators of Compromise  Help Desk/Non-security Incident Tickets  Risk Assessments  Penetration Testing Results  Application Security Scans  Firewall Port Requests |

Given the plethora and diversity of sources, your first and foremost task is to channel your inner-librarian to create and maintain a comprehensive catalog of these sources, *even if you’re not going to use them for analysis right now*. While some metadata will be unique, there are basic/common elements to record for each component:

**What is the generator of the data?** This could be as broad-based as identifying an organization-wide proxy server farm or as specific as the components of a line of business web application. Identifying the data source generator is important, though, especially as products and applications are retired or upgraded. Having a clear description of the source will make it much easier to make updates or changes when you perform reviews of your catalog.

**What is the actual or potential security purpose of the data source?** The main thrust of this question is to determine how the data can or will be used for security-oriented analyses. While there is a definite school of thought that security practitioners should “log all the things”, this is truly not practical even in the age of “big data” and cheap storage. We’ve used the word *potential* on purpose since there is a huge difference in having a catalog of all data sources and actually using them all. Think if it in terms of an inter-library book loan. The book is in the catalog and you know you can get access to the resource when you need it, even if it isn’t at your local library at the moment. Knowing what and where the data is can save a great deal of time later on, especially if you’re in the middle of an incident.

**Who is the owner/custodian/controller of the data source?** Here, you should be recording the contact information of at least two people entrusted with care and feeding of the data. This is usually the application/service owner and it’s a good idea to go refresh your catalog on at least an annual basis to ensure you have up-to-date contact information. These records will come in handy when you have access or processing issues (and, you will).

**What steps need to be taken to access to the data source?** Most data sources require special permissions to gain access to them and many have special or unique access methods. In some cases, this will involve pulling data via sftp, FTP, http[s], rsync, nfs or CIFS either in real-time (streaming) or at timed-intervals (batch).  You may also need to make direct SQL calls to a myriad of databases or REST3ful and/or SOAP4-based queries to retrieve data from more modern sources or proprietary appliances. Finally, it’s almost a guarantee you’ll be either a Windows event log consumer or syslog consumer and having a solid inventory will help you in building and scaling a good log management environment. Capturing solid details on the access method(s) will also be invaluable when it comes time to debug why data mysteriously stopped flowing into your analytics engines.

**What format(s) are the data elements encoded in?** In a way, your security analytics ingestion “hub” will be a digital Rosetta Stone5, knowing how to read and translate multiple representation formats into ones your engines can process. You can expect to be required to parse comma- and tab-separated (CSV/TSV) records, JavaScript Object Notation (JSON) objects, XML, Common Log Format (CLF) and a myriad of custom log formats. You may also be capturing raw packet capture (PCAP) dumps, SNMP traps or even unstructured text files. Recording this attribute will help you know when you need to add a new translator type to your repertoire.

**What mechanisms are in place to validate the integrity of the data source and transmission processes?** We’ve hinted that your data acquisition setup will be less than perfect and the answer to this question should help provide an early warning system when gremlins decided to creep into your processes. If a source should be generating a mean of “n” events per second and that suddenly drops down a few standard deviations or—even worse—to “0”, you can fairly confidently assume that this is something you should investigate, especially if it’s a more vital data source (say, firewall logs or IDS alerts). Similarly, you may be expecting to consume a data source that has eight fields per event that now has more or fewer fields as a side effect of a vendor “upgrade”. Having a process in place to validate and notify of such integrity issues can help prevent lost time and visibility down the road.

**What are the record retention policies for the data source?** If you’re not in a large or highly regulated organization, issues such as data retention tend to come up only when storage space becomes scarce. However, in many companies there are strict policies on how long you must maintain access to certain types of data. You should check with both the data source owner and your legal/compliance department to determine what your responsibilities are as a consumer and processor of the data. Many times, a data source owner will believe they have transferred responsibility to the security department without explicitly stating so up front. Getting this confirmation can stave off future headaches and potential legal issues.

**Where, physically and logically, is the data?** Having an understanding of the geographical and network locations of the data source will be very helpful when it comes to actually implementing the data intake procedures. If you have identified that a source is in a Colorado data center but your collection and analytics engine is in the Detroit data center, you know that there are potential availability, capacity and latency issues you may need to deal with when building out and maintaining your intake and analytics engine. Ideally, this information will be obtained from your existing asset management system, since your security analytics inventory should—ideally—be a reference source, not the authoritative source.

**What is the expected volume (*how much*) and velocity (*rate of transmission/expected consumption*) of the data?** You may say you want real-time access to your organization’s firewall logs but that access will do little good if your processing engines are not capable of digesting that fire hose of information. Thankfully, it’s fairly straightforward to measure and estimate the volume/velocity of the data flow and design an appropriate intake configuration. Skipping this step may end up with you only retrieving only one out of every four events and missing potentially critical data elements or worse, launching your own denial of service (DoS) attack on your logging and processing servers.

This foundational catalog can provide fertile ground for formulating insightful research questions, identifying elements to track for your metrics program or aiding in incident response activities. Whether you’re asking a question only once or generating a weekly report, you’ll need to start collecting real data to process.

Building Your Data Intake And Analytics Engine

Now that you have an idea of the data sources that are available and at least an initial estimate at which ones you’ll be using, it’s time to build the supporting collection and processing systems. If you think of the data as fuel, your goal here is to design the most efficient way to get that fuel into your tank for storage and engine for processing. This will not be a one-time event as you will always be incorporating and potentially retiring data sources and developing new ways of crunching the data. Figure 3-1 provides a conceptual overview of this data flow.

Figure 3-1: High Level Overview Of Data Intake/Processing Flow [f0301.eps]

Keeping the intake process as straightforward as possible should be your primary goal as that will make it much easier to diagnose and fix issues as they crop up and will also simplify what mechanisms you need to put into place to get at this data for analysis.

First, you must decide what your system, network and application event logging strategy will be. The concept of a homogenous data center environment is but a faded memory—if it ever truly existed—since even the most Windows-centric organization cannot help but have its share of routers, switches, firewalls and appliances, not to mention a mainframe or Linux box (or two). Despite the continued prevalence of syslog-based logging across an ever increasingly wide spectrum of systems, applications and devices, Windows systems still hold hard and fast to using Windows event logs as their preferred log language (much like how the French fight the linguistic hegemony of English). Given this data center dichotomy, you will need to make a choice as to whether you’re going to use a solution like Microsoft System Center Operations Manager (SCOM) to handle all events—including, using their syslog adapter for everything besides Windows components—or whether you will be using a logging translator in your Windows environment to get events out into a more common tongue.

Larger organizations with equally as large budgets may benefit from a proprietary log management solution such as HP ArcSight Logger or Splunk since any engagement involving tools like those will also include some consulting to determine what your needs are. However, it’s possible to roll up your sleeves and architect a robust open source collection hub with something as traditional as syslog-ng6 (since you can even forward Windows events to it using a tool like Snare7) and more modern tools such as logstash8 and Graylog29. Figure 3-2 provides a more detailed picture of this part of the intake hub.

Figure 3-2: Expanded Event Logging View Of Intake Hub [f0302.eps]

Keep in mind these three core, guiding principles for your intake hub:

* It must have the ability to consume feeds/events from current and future sources. This means it must be scalable and flexible.
* It must have the ability to store feeds/events for later retrieval/processing. This means the storage must also be scalable.
* It must have the ability to stream feeds/events, possibly translating them into a more usable format, into your analytics engines.

Observant readers will notice the lack of reference to Hadoop. While you could jump right over to Chapter 7 to take a peek at *where* Hadoop *can* fit into this configuration, it is not—as we’ll see later—the singular, ultimate solution to all your logging and processing needs.

The previous section introduced the concepts of data *velocity* and *volume* because you will need to architect your environment well enough to handle the entire incoming flow of data from each source and scale up and out as you add new ones. You may be used to the term *velocity* being referred to as the number of *events (or messages)-per-second* and *volume* as the aggregate *event record (or message) size*. Here is where your systems administrator powers will come in handy. NOTE: You may also end up needing to exercise your budget bending powers as well, since this may be a more expensive task than you might, at first, think.

As is the case with most computing environments, your ability to intake and crunch through data is limited by:

* Network interface capacity
* Disk speed/capacity
* CPU capacity
* Memory

There are many tomes dedicated to capacity planning, but here are a few rules of thumb to keep in mind when building and scaling your security analytics environment:

**Partition/distribute workloads**. Unless you’re dealing with a small environment or are severely fiscally constrained you should not expect a single server to handle all your logging and computational needs. Let loggers log and search/analytics nodes analyze and design each of them to scale both for data capacity and human capacity, since if you build a robust security analytics environment, it *will* attract users. Some technologies, such as MongoDB and Hadoop have clear requirements on how they must be scaled, especially if you have data integrity and availability requirements to meet. Other use cases, such as the intake and storage of feeds/logs have much less demanding constraints. You’ll need to understand each service to determine how best to arrange the components.

**Calculate and use standard deviations**. Workload estimation is one of the darker arts of computing and can be a bit daunting at first, but definitely becomes a bit easier over time as you gain experience and familiarity with the environment and processes. While you’re on that journey, doing some basic analysis on the sources feeding your environment to understand the mean (average) volume and velocity and what 2-3 standard deviations out look like will provide a good starting point for initial capacity requirements and applying a bit of linear regression analysis (Chapter 5) over time will help you forecast expansion needs.

**Optimize opportunistically**. Even though the computing power and storage costs are relatively cheap, they aren’t free and you should not be looking to preemptively optimize components of your setup until you are sure of both the bottleneck and what will help overcome it. For example, stocking your setup full of the fastest solid state drives (SSD) can set you back–as of this writing–close to $16.00USD/GB. However, a combination of RAM and higher end regular drives may be sufficient, if all you’re trying to do is provide more capacity for incoming logs. Neither will help if you’re consuming all available network bandwidth on the system, so it’s important to have a reliable performance-monitoring plan in place to make it more straightforward to diagnose and remediate issues.

[ORCHESTRATING THE ENVIRONMENT WITH SALT]

[TALK ABOUT HOW EACH SOLUTION WILL BE UNIQUE]

Using A Code/Scripting Repository

The whole purpose to setting up a storage and analytics environment is to actually run queries and analyses on the data. The code you write to perform these functions and keep the system humming along are as vital to you as a toolbox is to a mechanic. The most effective “wrenches” know precisely where their tools are and ensure they always have the right ones on hand for the job. One of the best ways to mimic this practice in the digital realm is to use a revision control system (RCS) that enables multi-user/distributed update and retrieval of the code that powers your data collection and analytics with the option to . Rather than constraining your creativity, RCS environments can actually help foster even greater experimentation since you are moving your code hacking skills into a more structured—but not constrained—process where it’s far more difficult for one random edit to ruin a script or force a retrieval from backup.

Whether you use—or are required to use—proprietary solutions such as IBM’s Rational tools [REF:] and Microsoft’s Visual SourceSafe [REF:] or open source options such as git [REF:http://gitscm.com/] or Mercurial [REF:http://mercurial.selenic.com/], learning the basic functionality of and standing up an RCS environment will be a fairly quick task that will reap many benefits as you use it, not the least of which is making your research and analytics *repeatable* and *reproducible* (more on that in the next section).

[PRINCIPLES]

**Work locally, store “remotely”.** Most modern RCS tools give you the freedom to create and develop in a local repository and then publish and update changes to a “remote” (centralized) server that can act as a backup environment, distribution server or collaboration hub. Your workflow will be slightly altered, but you can continue to use your favorite editors and tools in between committing your changes.

**Branch often, merge judiciously.** The term “branch” means to veer from the stable, main development path to experiment without wrecking anything. This can come in handy if you’re itching to try out a new library, idea, snippet or technique but aren’t sure if it’s going to work as intended. Sure, you could do this with regular file system commands, but using an RCS will enable you to seamlessly merge successful experiments back into the main code and still provide a means of going back to “safe” or alternate revisions at some point down the road.

**Build skeletons, clone**

Notes

1. Joseph F. McDonald, “Russell, Wittgenstein, and the Problem of the Rhinoceros”, *The Southern Journal of Philosophy* (Volume 31, Issue 4, Winter 1993; pp.409-424)

2. http://en.wikipedia.org/wiki/Captain\_Marvel\_(DC\_Comics)

3. http://www.ibm.com/developerworks/webservices/library/ws-restful/

4. <http://www.ibm.com/developerworks/webservices/tutorials/ws-understand-web-services1/section2.html>

5. <http://en.wikipedia.org/wiki/Rosetta_Stone>

6. <http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/>

7. <http://www.intersectalliance.com/projects/SnareWindows/>

8. <http://logstash.net/>

9. <http://graylog2.org/>

http://bost.ocks.org/mike/make/

Recommended Reading

*Mastering Unix Shell Scripting: Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and UNIX Gurus, 2nd Edition* by Randal K. Michael (Wiley, ISBN: 978-0-470-18301-4)

*The Art of Capacity Planning* by John Allspaw (O'Reilly, ISBN: 978-0-596-51857-8)

*Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services* by Neil J. Gunther (Springer, ISBN: 354031010X)