Stupid Log Tricks

A technical paper prepared for presentation at SCTE TechExpo24

Matt Carothers
Senior Principal Security Architect
Cox Communications
matt.carothers@cox.com

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

1. Introduction

Many years ago, a security team participated in a compliance audit. To pass the audit, they demonstrated to the auditor that their team maintained a full year of security log data. As proof of compliance, they sent a screen shot from their Security Incident and Event Management (SIEM) system showing it configured to store one year of logs. They passed the audit with flying colors, but in the best example I know of demonstrating "compliance is not security," they later discovered the SIEM only actually had enough disk space for thirty days.

Thus began a journey to upgrade or replace the SIEM, and they quickly realized they had fundamental architectural flaws. First, expanding storage would be expensive because the team relied on a single vendor using proprietary technology. Additionally, every device producing logs sent them directly to the SIEM and thus would require massive effort to reconfigure if a new vendor was chosen. Finally, the SIEM itself did not meet every need of a modern security team. While it alerted on real time events relatively effectively, searching for historical data for investigations or hunting could take hours to complete.

To solve those problems, the team started from a blank slate to create a new kind of logging architecture. It would be fast, flexible, scalable, and cost effective. This paper takes the reader from our original design with all its flaws to our modern implementation and its numerous benefits.

2. Logs! What are They Good For?

So why do we even need logs, and what does a SIEM actually do? At its core, a SIEM must fulfill two basic functions:

- **Real time detection**. It must examine a stream of log events as they arrive, correlate them with other events if necessary, and produce alerts for suspicious behavior.
- **Forensic search.** In addition to alerting, a SIEM must support investigation and threat hunting. When an alert fires, a cyber defense team must examine events leading up to the event to determine if an incident occurred. When a team receives new threat intelligence, it must search months or even years into the past to determine if a security event took place.

3. Beyond the Basics

A high functioning security team does not just need the basics. It needs a system that goes above and beyond to help the team excel. At our company, we designed our new architecture to meet the following additional requirements:

- **Cost-effective scalability**. Proprietary solutions can be very expensive to scale. Even if the underlying technology is open source, a customer pays for the brand name. We needed our system to be cheap and easy to expand.
- **Extreme flexibility**. Security tools come and go. The best solution for a problem may be a proven technology, or it may be a bleeding edge experimental technique. During a security incident, a cyber defense team may need to implement cutting edge detection systems on the fly while fighting head-to-head with adversaries. If every security log disappears into a proprietary system, the events cannot quickly be funneled to other tools. Furthermore, the extreme difficulty in changing SIEM vendors creates a disincentive to explore competing products. We designed our system to quickly and easily direct log events to multiple destinations.
- **License cost management**. SIEM vendors typically adopt one of two licensing models. They either charge customers based on the amount of data ingested or the number of events per second. Security

teams must often choose between throwing away logs that might be useful later or simply eating the cost. Our system allows us to manage costs by sending only the most necessary events to our SIEM while keeping the rest in cheaper systems.

- **Normalization and distillation**. Every cyber defender experiences the frustration that comes with wildly varying log formats. Each source produces different data with different field names, making it difficult to create a unified view of a security incident. Compounding the problem, defenders often must "swivel chair" between multiple tools, requiring them to copy and paste between different systems. Many must resort to tedious spreadsheets to understand the full picture. Our system seeks to both centralize and normalize all relevant data to give defenders a "single pane of glass" in which to work, while distilling events from raw data to useful knowledge.
- **Speed**. When a cyber defense team receives Indicators of Compromise (IOCs) such as Internet Protocol (IP) addresses, domain names, or file hashes, it should be able to get an instantaneous yes or no as to whether the IOC has been seen in the defender's network. Waiting minutes to hours for a query to return results cripples a cyber defender's ability to respond quickly to rapidly changing information.

4. Architecture Overview

4.1. The Old Way of Doing Things

Our previous architecture consisted of devices such as routers, firewalls, and servers sending log messages directly to a proprietary SIEM.

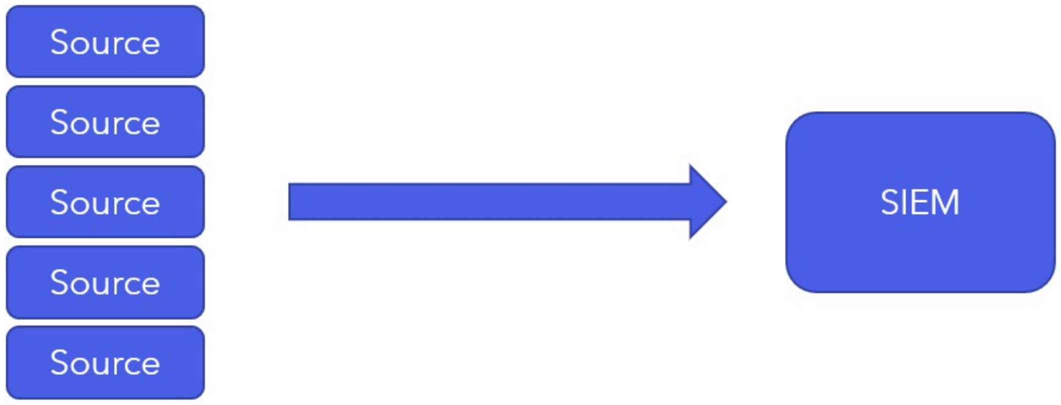


Figure 1 - Previous Architecture

While this architecture did serve to provide real time alerting, it failed on most other counts. Historical data searches took hours, upgrading the platform was prohibitively expensive, and changing platforms or allowing other tools to utilize the log data would have required configuration changes on thousands of devices.

4.2. The New Architecture

Our new architecture inserts a broker built on open-source tools between our log sources and destinations. The broker not only multiplexes our logs, sending copies to multiple destinations, but also provides filtering, enrichment, and transformation.

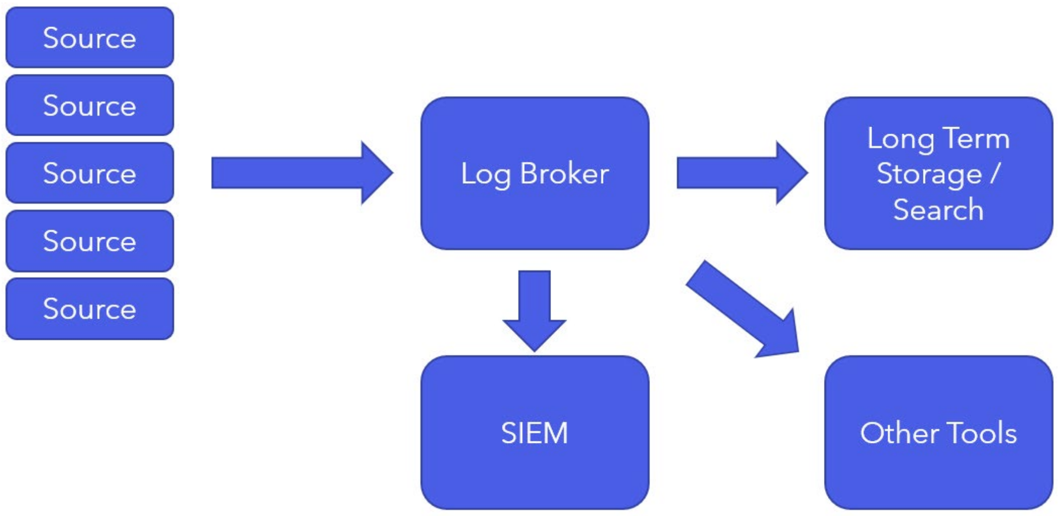


Figure 2 - New Architecture

It immediately provides flexibility by allowing us to split the traditional SIEM functions of real time detection and forensic investigation into separate best-of-breed tools. Additionally, should we decide to replace our SIEM, we can simply stand up a new product in parallel to the old one and send copies of the log data to both products until we are ready to turn the old one down.

5. The Log Broker

We refer to our broker as a “Logstash sandwich.” Logstash[1] is the Swiss Army Knife of logs. It ingests data from a wide variety of sources, transforms it, and delivers it to many destinations.

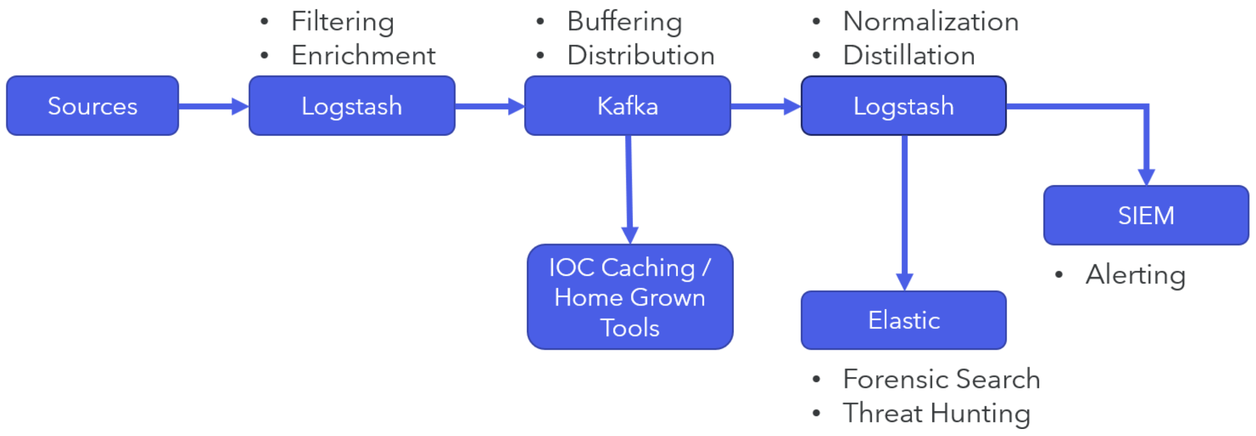


Figure 3 - The Logstash Sandwich

5.1. The First Logstash

The first Logstash cluster, fronted by a load balancer, which is not pictured in the diagram, receives logs, mostly in syslog format. Its job is to receive data as quickly as possible without losing any packets. Additionally, it forms our first line of defense against unnecessary data that can be immediately dropped,

[1] https://www.elastic.co/logstash

and it provides simple enrichment, such as geolocation. The cluster scales quickly and cheaply simply by deploying additional Logstash instances.

5.2. Kafka

After processing the incoming logs, the first Logstash delivers its data to a Kafka[2] message queue. The queue serves two purposes. First, it provides a buffer to guard against cases when the second Logstash layer might be overwhelmed or unavailable. Second, it provides an additional method to get data into and out of the system for sources and destinations not supported by Logstash. For example, we receive logs from sources such as Azure Event Hub by making API calls and inserting the events directly into Kafka. Homegrown tools discussed later in this document also pull from Kafka.

5.3. The Second Logstash

The second Logstash cluster pulls data from the Kafka queue, transforms it, and delivers it to multiple locations. We deliver a subset of logs to our SIEM for real time alerting and multiple copies in different formats to an Elasticsearch stack for forensic investigation.

6. Meeting Our Goals

- **Real time detection**. In addition to delivering logs to a traditional SIEM, our architecture allows us to use home grown tools that provide targeted detections not supported by the vendor. For example, one such system keeps track of every IP address an employee has logged in from and performs automated investigation when an employee logs in from a new one.
- **Forensic search.** By delivering logs to Elasticsearch, we gain a forensic investigation tool designed specifically for scalable searching. We are no longer tied to whatever storage system our SIEM vendor designs or white labels. By utilizing a mix of both commercially-supported and open source Elasticsearch clusters, we store petabytes of data while largely only making one-time capital expenditures for off-the-shelf servers with large disk drives.
- **Cost-effective scalability**. Everything in our stack scales horizontally with ease, and most of it is open source, requiring no licensing costs.
- **Extreme flexibility**. With multiple paths in and out of the system, utilizing Logstash's impressive array of input and output modules along with our own code, we can accept data from any source in any format.
- **License cost management**. Because we only deliver logs to our SIEM that directly relate to detection use cases, we reduce the volume by approximately 90%. It allows us to keep our licensing costs down without having to sacrifice data that might later become valuable.
- **Normalization and distillation**. Logstash's extensive toolbox of transformation functions allows us to deliver data in multiple formats. We normalize our data using the Elastic Common Schema (ECS) and utilize a technique we call "distillation" that reduces a log to its essential meaning. We will discuss those in depth later in this document.
- **Speed**. Our architecture allowed us to build a Redis-based caching layer called the Observed Indicator List (OIL) that stores the last known log entry relating to any given IOC. It gives us an instantaneous yes or no answer as to whether we have seen a given IOC in our environment as well as a pointer to the most recent occurrence.

[2] https://kafka.apache.org

7. Normalization

Anyone who has dealt with security logs has experienced the frustration that comes with different formats. Although most log sources contain common elements, such as IP addresses, usernames, and email addresses, the field names often vary. A cyber investigator often needs to have specific knowledge of a particular log source to know which field names to search, and producing unified dashboards covering multiple sources becomes complicated if not impossible.

To address this issue, we normalize our logs using the Elastic Common Schema (ECS).[3] ECS provides an open-source reference for naming various security information. For example, the field name for a source IP address is "source.ip."

For example, consider the following formats:

Source	Log Message	IP Field
Windows	EventID 4624: An account was successfully logged on. SourceIP: 192.168.1.100	SourceIP
DHCP	[DHCPIP: 192.168.1.100] to MAC address 00:1a:2b:3c:4d:5e Tuesday, Jan 16	DHCPIP
VPN	Jan 16 12:18:06 User connected from IP: 192.168.1.100	IP
Email	Jan 16 12:18:21 dovecot: pop3-login: Started proxying to <127.0.0.1> (0.020 secs): user=<#@####>, rip=192.168.1.100, lport=995	rip
Okta	{"displayMessage": "User login to Okta, "published": "2024-01-16T12:24:02.897Z", "ipAddress": "192.168.1.100"}	ipAddress

Figure 4 - Example Log Formats

A query covering those sources would look something like this:

```
SourceIP: 192.168.1.100 OR DHCPIP:192.168.1.100 OR
IP:192.168.1.100 OR rip:192.168.1.100 OR ipAddress:192.168.1.100
```

By normalizing logs to ECS, we can query them all with a single field instead:

```
source.ip:192.168.1.100
```

8. Distillation

We use the term "distillation" to describe our process of extracting meaning from security logs. We take a raw log, such as a web request to a particular page, and distill it down to an essential meaning, such as "a customer logged in from a particular IP." Distillation greatly reduces the amount of data we store, increases the time we can retain it, and allows us to easily integrate it into cross-source dashboards to produce a unified timeline of a security event.

[3] https://www.elastic.co/guide/en/ecs/current/ecs-reference.html

For example, consider the following raw log message:

```
<134> hostname12345 2024-01-18 15:53:08,192 GMT DEBUG
[Format=TRIAGE|Class=com.cox.oss.core.servlets.TransactionFilter]
(http-nio-8080-exec-
215|E:asdfghjksdfsdfsdfd|R:asdfsdfdffgfdgfgj|V:cox|C:Test_Applicatio
n|ThreadId=11167050) Responding [PUT
/residential/identities/2.1/guid/111111111-222222-333-444-
555555/mfa/factors/SMS/verify] with outbound response in 452ms: HTTP
200 X-Cox-Request-Id: asdfsdfdffgfdgfgjj X-Cox-External-Id:
asdfghjksydfsdfsdfdfsdfdfffd {"results":{"customerGUID":"111111111-
222222-333-444-
555555","userLoginInfo":{"passwordChangeDate":"***MASKED***","passwo
rdChangedBy":"***MASKED***","identityCreatedBy":"abcdefgh","lastLogi
nTime":"2024-01-
13T22:38:53.000Z","userName":"test_user","locked":false},"mfa":{"enr
olled":true,"forced":false,"factors":[{"id":"sfsdfsdfsdfsdfdffsd","f
actor":"SMS","factorValue":"+12345678910","status":"ACTIVE"}]},"auth
orizations":["ACCOUNT","WIFI"],"primary":true,"accountInformation":{
"accountGUID":"abcdefghijklmnop-dfeg-2vfd3-sdfd-
sdfsdfd","statusCode":"A","billType":"S","accountServices":["WIRELES
S","VIDEO","DATA"],"serviceAddress":{"streetAddressLine1":"Test
Address","streetAddressLine2":"123 Fake
Street","city":"Atlanta","state":"A","zipCode":"90210","timeZone":"A
merica/New_York"},"accountNumber":3456789,"secretQuestionCode":"A2",
"residentNumber":11,"site":222,"sixteenDigitNumber":"112345678910111
2","houseNumber":123456,"thirteenDigitNumber":"1234567891012","twelv
eDigitNumber":"123456789101","company":11,"division":2,"active":true
},"appPasswords":[]},"version":{"application":"idm-profileservices-
2.26-
GA","api":"2.1","buildNumber":"270"},"executionTimeInMillis":452,"in
ternalTransactionId":"f","externalTransactionId":"sdfsdfsdfsdfsdfsdf
sdfsdfsdf","timestamp":"2024-01-18T15:53:08.192703211Z"}
```

It contains a great deal of extraneous information and tells us very little about what happened. To extract meaning, we first normalize the log using ECS:

Field	Value
@timestamp	Jan 22, 2024 @ 19:34:48.963
http.request.method	PUT
http.response.status_code	200
IdmProfileServices.Source_Application	Test_Application
url.path	/residential/identities/2.1/guid/111111111-222222-333-444-555555/mfa/factors/SMS/verify

user.target.ICOMS_Account_Number_9	003456789
user.target.ICOMS_Site_Id	222
user.target.name	Test_user
user.target.MFA_Factor	12345678910
user.target.MFA_Factor_Type	SMS

Figure 5 - Normalized Log Information

While normalization cleans the entry up and standardizes the field names, it does not tell us what the log means. In this example, the log represents a customer enrolling their account in Two Step Verification (TSV), so we distill it like this:

Field	Value
@timestamp	Jan 22, 2024 @ 19:34:48.963
event.action	sms_tsv_enroll
event.outcome	success
event.Friendly_Name	SMS TSV Enrollment
event.Description	Customer identity Test_user enrolled SMS MFA factor 1234567891 via Test_Application: success
user.target.name	Test_user
user.target.MFA_Factor	12345678910

Figure 6 - Distilled Log Information

Combined with distilled logs from other tools, we can quickly assess and communicate what happened during a security event, even when the logs span multiple sources.

9. Stupid Log Tricks

We can detect security incidents in real time. We can investigate and hunt through historical data. But what else can we do with logs?

9.1. Passive Asset Inventory

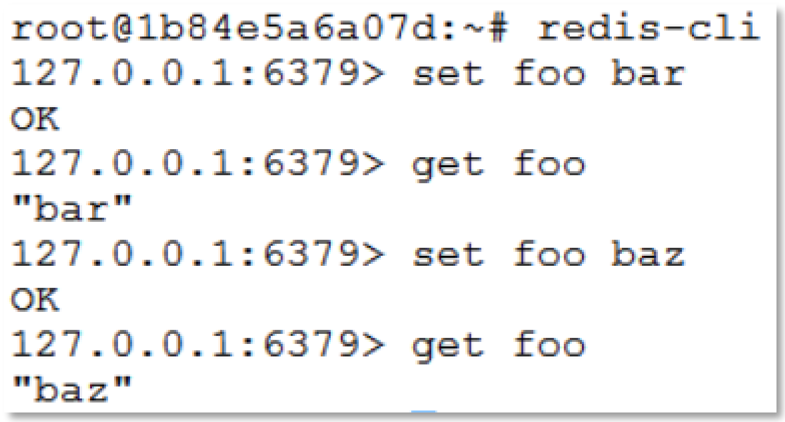
Incomplete asset inventories are the bane of incident responders everywhere. An alert fires, but no one knows what the IP represents. The defender is unable to determine what the asset is or who owns it. But wait! We can gain a great deal of insight from security logs.

- Having a log from a device usually tells us what type of device it is. We immediately know whether it is a Windows or Linux server or some other type of hardware. The logs often contain other information as well, such as the device's configured hostname.
- Sign in logs tell us the identity of an account with access to a device. If we do not know who owns it, we at least know one person who can log in. If that person is not the system's owner, they almost certainly know who is.
- Firewall and Endpoint Detection and Response (EDR) logs give us similar information. For example, if firewall logs show a user connecting to a host on port 3389 (Remote Desktop Protocol), we can infer the destination is a Windows server and the source user has access to it.

9.2. The Observed Indicator List (OIL)

Threat intelligence teams receive tens (if not hundreds) of thousands of IOCs every year. The vast majority will not be found in the defender's network. Searching for every IOC across petabytes of data in a traditional database is slow at best and completely infeasible at worst. Thus, rapidly answering the question of what has not been seen is of paramount importance.

To understand OIL, we must first discuss Redis.[4] Redis is a key/value store. It is a specialized database without tables, columns, rows, or other traditional data structures. Instead, it allows a user to store a value into a key and retrieve it later. Setting a new value into a key overwrites the old value.



```
root@1b84e5a6a07d:~# redis-cli
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> get foo
"bar"
127.0.0.1:6379> set foo baz
OK
127.0.0.1:6379> get foo
"baz"
```

Figure 7 - Redis Example

This type of database does not get slower as more keys are added. No matter how many keys are set, retrieving a value still takes mere milliseconds. The data remains fully in memory but can be written to disk for persistent storage.

Now that we understand Redis, let us talk about OIL. The concept behind OIL is simple, but powerful. For every IOC contained in a log message, we assign a key. The key is the IOC itself, and the value is information from the message.

For example, consider this Azure sign-in log with IOCs highlighted:

[4] https://redis.io

```
{
  "@timestamp": "2024-06-28T22:03:33.854890215Z",
  "source": {
    "Device": {
      "Name": "DESKTOP-H8YS09"
    },
    "ip": "1.2.3.4",
    "user": {
      "email": "john.doe@company.com",
      "full_name": "John Doe",
      "name": "jdoe"
    }
  }
}
```

For this example, we would create four Redis keys:

- DESKTOP-H8YS09
- 1.2.3.4
- john.doe@company.com
- jdoe

The value assigned to each key would be the log message itself. If in the future the team receives threat intelligence indicating 1.2.3.4 is hostile, a defender can "check the oil" to find the most recent login from that IP.

Other examples of OIL sources include the following:

- Netflow – keys are the source and destination IP
- Firewall logs – keys are the source and destination IP
- Sign in logs - keys include the source IP, hostname, and usernames
- Asset database – we export our asset database into OIL to provide instant lookups within the same tool used for IOC lookups

10. Conclusion

In conclusion, inserting a broker between log sources and the tools that consume them provides scalability, cost savings, and the flexibility to innovate. Normalizing and distilling the data makes it easier to search and easier to create a holistic view of a security incident. Applying some creativity to log data gives defenders new capabilities, such as creating an asset inventory without needing input from other teams. Finally, using the OIL technique to cache IOCs allows cyber defenders to find the needle in the haystack by instantly removing all the hay.

Abbreviations

API	Application programming interface
ECS	Elastic Common Schema
EDR	Endpoint Detection and Response
IOC	Indicator of compromise
IP	Internet Protocol
OIL	Observed Indicator List
SIEM	Security incident and event management
TSV	Two Step Verification