17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 129

Navigation System Analysis: Cost Design and Implementation of a Security Event Logging Framework.

R. M. Beswick[a]*

[a] *Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109, USA,*
Robert.M.Beswick@jpl.nasa.gov
* Corresponding Author

Abstract

This paper examines design approaches and techniques for implementing a low-cost Security Event Management (SEM) logging framework based on our experience deploying such systems on a mission-critical network. This deployment was a crucial part of the effort to implement and protect mission-critical flight systems used by the Mission Design and Navigation Section at NASA's Jet Propulsion Laboratory (JPL). Spacecraft navigation is the result of a complex interplay between numerous teams, performed on JPL's Multi-Mission Navigation (MMNAV) Ground Data System. This paper reviews the process used to design fault tolerance and robustness into the security architecture, the goal of which was to protect the users of the system from hostile external threats, while making their experience on the system as user-friendly and efficient as possible over the entire lifecycle of the system. Deriving from the fault tolerant design concepts of *fail-gently* and *independent failure*, this paper particularly focuses on the implications of examining and protecting the evolution of this system over time. Viewed especially against the backdrop of principles seen from *Time Based Security,* the design goal was a robust system that is secure from its implementation to its disposition. From this clear need for a secure, robust, high-reliability/high-availability computational environment to support navigation data processing, our team designed Security Information Event Management (SIEM) into the MMNAV Ground Data System. From system requirements, user, and administrator feedback, we developed a model for SEM for the overall system. Working with open-source software, an iterative approach was employed to capture and observe user actions and data flows, store such logs in a secure manner, and then implement a system to review and monitor such logs accordingly in real time. The goal was to have a means to examine the current security state of the entire MMNAV Ground Data System as it evolved over time. While numerous security vendors will provide such services or appliances at a price, we desired an affordable system under our control, one that we designed and understood from the ground up. This paper discusses the challenges of implementing this SIEM in a low cost, but robust framework – using the open source tools of *Syslog-ng*, *Elasticsearch*, and *Kibana* to create a system allowing real-time monitoring of our secure ground data system. Hopefully this can serve as an excellent example for future missions with similar needs.

Keywords: Computer Security Engineering, Ground Data System, Time Based Security, Operations Security, Security Fault Tolerance, Complementary Intersection

Nomenclature

Pt	=	System Protection, time required to break through
Dt	=	Detection time
Rt	=	Reaction time
t0	=	Initial state of a system (at start time)

Acronyms/Abbreviations

CIA:	Confidentiality, Integrity, and Availability (foundational security principles)
CM:	Configuration Management
DR:	Disaster Recovery
Elasticsearch:	A distributed search engine with a web interface
GDS:	Ground Data System
JPL:	Jet Propulsion Laboratory
Kibana:	Data-visualization software set for Elasticsearch
Linux:	Open-source OS derived from Unix, System V-based OS
OS:	Operating System
MTTR:	Mean Time To Restore
MMNAV:	Multi-Mission Navigation (operations coordinating organization)
QoS:	Quality of Service
RHEL:	Red Hat Enterprise Linux – "Enterprise class" commercial Linux distribution
SEM:	Security Event Management
SIEM:	Security Information Event Management
SFOC:	Space Flight Operations Facility – Mission-critical spacecraft operations building at JPL
SOC:	Security Operations Center – security team monitoring an organization's IT systems
SSH:	Secure Shell communications replacement for RLOGIN, RCP, and other "R" commands

1. Introduction

This paper is an examination of the approaches developed to fit a particular need in securing the Multi-Mission Navigation (MMNAV) Ground Data System computer systems. This Navigation Ground Data System comprises several networks supporting the navigation elements of more than forty interplanetary missions here at the Jet Propulsion Laboratory. This computational environment is constructed in the manner of a development and operations model with multiple networks of several hundred servers and workstations. This discussion is a follow-up to previous papers that examined the overall design, implementation, and operations of such systems [1, 2] and the process of securing those computer systems [3, 4, 5] with an eye to considerations of resiliency and security-fault tolerance for such systems. In particular, this paper moves away from the previous discussion of the static implementation of a secure computational environment to consider a demonstration approach for monitoring that environment as it operates. This paper considers an example of how an inexpensive and efficient setup was implemented and deployed, giving our system engineering and administration team at JPL the ability to observe the network security state in real time and monitor for events and specific patterns of occurrences.

This effort took place against in the world of aerospace that systems administrators and systems engineers regularly contend with. It is a world of often austere budgetary constraints, with few systems requiring as much resiliency as the computer systems supporting flight operations, where failure is often measured in the loss of irreplaceable space-based hardware and/or immeasurable lost scientific returns. With scant resources available for the maintenance of complicated architectures of systems and software, considerable care must be taken and resources used wisely.

With this in mind, this paper gives a brief overview of the process of designing a secure system, and then moves to consider, in the context of *security over time* (or robustness), how one protects that system as it evolves over time. As a part of this protection, an effective means of monitoring the interconnected networks and computer systems – a process for Security Event Management (SEM) was implemented to allow for not only static analysis of prior event data, but also customizable analysis of data in real time.

Efficient implementation of such a framework, in the overarching context of secure system design, is the subject of this paper. The goal here is to give other teams of systems engineers and systems administrators assistance in their own computer networks secure as they evolve over time. This document presents a solution involving a set of tools and approaches that will enable a small team with limited resources to monitor security events in a simple, easy-to-architect fashion. This framework is meant to add a few tools to the metaphorical toolbox of the systems implementor. To be very clear, this is not meant to replace the full capabilities of a dedicated security team at a Security Operations Center, nor is it intended to be an exhaustive examination of the full range of how such systems may be set up. Such concerns require a much wider discussion, which greatly exceeds the limited scope of this paper. Indeed, some credit and the initial inspiration for this work came from the author's attendance at a two-day seminar at a SANS conference in 2018, where he sought to distill a survey of industry best-practice approaches into one or two usable mechanisms for the MMNAV system engineering and administration team [6].

1.1 Static System Design Overview

The goal of these considerations is the design and operation of a secure computation environment. As noted above (in 1.0), this has been covered in previous papers, so only a brief overview will be provided here. We start out by implementing a security fault-tolerant computational environment representing a best-effort compromise between the operational needs of the user community and that of the desire for a solid security implementation. ([5] is a detailed discussion of this.) This is our initial state, our *t0*, that we will then protect over its entire lifecycle – and we want to make it a good one, as strong fortifications are easier to defend than weak ones. The following subsections describe basic paradigms used with this design:

1.1.1 Security Fault Tolerance

Designing in security-fault tolerance in a system is akin to general fault-tolerant design in that crucial techniques lie in avoiding single points of failure of a system; having redundancy of critical components; and using components that, when they fail, undergo graceful degradation of their capabilities (also known as *fail-gently*) [7]. Additionally, such components should have the ability for independent failure and independent *modes* of failure (i.e. differing types of failure) [8]. With the consideration of security-fault tolerance however, such systems are not only resilient from random chance failures but also resistant to intelligent direct action [9].

Implementing security fault tolerance revolves around the following techniques [10, pp. 360-363]:

1) For the most critical processes, use a system that does only one function.
2) For redundancy in a security context, differing configurations (in redundant systems) should be used.
3) Single points of failure of a system should be few and independent of each other.
4) System failure modes should *fail-safe* and / or *fail-gently*.

1.1.2 Computer Security Design

These techniques then are combined with the basic principles of computer security and secure implementation. The abstract principles of computer security, known as *Confidentiality*, *Integrity*, and *Availability,* or *CIA*, are used to determine where to concentrate and apportion resources depending on the needs of the end users and job tasks (e.g. a computer system in use by an intelligence agency may have very different needs and concerns than one dedicated to space exploration for public scientific research) [11, pp. 4-6]. Using these principles, a number of implementations are used:

1) Defense in Depth: Overlap defenses used so that failure of one part does not lead to failure of the whole.
2) Defense in Breadth: Ensure protection of the whole system that may be vulnerable to attack.
3) Least Privilege: Users are given only the access needed for the tasks they need and no more.
4) Vulnerability Removal: Remove access to the system by methods where such access is not needed.

1.1.3 System Evaluation

Using computer security design approaches with techniques of security-fault tolerance, a system may then be evaluated to produce an optimal start case or state. Such an optimal condition must include consideration of the use of the system by its intended users and their operational concerns. As discussed in the citations in Section 1.0 this optimal beginning state is evaluated at different areas, with different optimization points for such areas as the external network access, internal host security settings, and user access controls.

Using these approaches and techniques, we carefully restrict access in key areas. For example, we restrict the external network-facing points (akin to locking the front door), while allowing a bit more freedom in the internal host settings and user accesses (akin to allowing people to move around freely inside a house). Once the system setup is finished, the system is at its maximum state of security. This is the beginning of the security life cycle, the initial starting point or *t0*.

This lifecycle has been likened to issues seen in medical care, or food handling and preparation, as *Sterility in Implementation* – namely that once this system is handled or accessed such sterility is compromised and its state begins to evolve over time (usually not in a positive fashion) [5].

1.2 Security Over Time or Robustness

As we consider this evolution over time, this security lifecycle, questions such as “how robust is this system?” arise. How much can it be trusted and how does that trust change over time? This is strongly dependent on the environment and connections to the system. It is important to examine these connections and methods of contact to characterize the *Security Over Time* of a system [12, pp. 102-105].

These considerations follow closely along with the concept Winn Schwartau called *Time Based Security* where he makes the analogy of the design of a secure bank vault, with its alarm system [13, p. 29-40]. In this concept, a system

would be considered secure if the time needed to break through the system protection (Pt) was greater than the time needed to detect (Dt) and respond (Rt) to the intrusion:

$$Pt > Dt + Rt \qquad (1)$$

There is considerable difficulty in quantifying the values in (Eq. 1) for the system protection, detection, and reaction time in the system. [12, pp. 45-46] (This is analogous to the fault-tolerant concept of *fail-gently* as discussed in Section 1.1.1, as well as several other engineering reaction time-based analyses – including that of braking distance for a vehicle).

As we consider the evolution of a system state over time, how this state changes is subject to the environment that the system operates in. These changes in confidence may be nearly linear over time or may be an exponential die-off thanks to newly discovered vulnerabilities. Such factors as software sets, especially in network-connected software, security controls in place, and other environmental factors acting on the system, can give differing systems on the same network wide variance in how they evolve. Indeed, in the worst-case scenario, a system compromise will cause the system to have a cliff (stair-step) decline in the confidence of the system.

1.3 Specific Approaches for Time Based Security

The one component in Eq. 1 that can be varied easily is the detection time (*Dt*) for a security event. Up to this point we have considered the whole of the security challenge. Now let us consider specific means to reduce that detection time. SEM works hand in hand with the more focused concept of Security Information Event Management (SIEM). While systems are configured as a part of our static system evaluation process to generate security logging and auditing data, including sending such security events to secure remote servers, being able to review such logs and events in real time in a configurable fashion is a greatly superior kind of process. This SIEM process, (and the concurrent decrease in detection time) is enabled by better and more efficient means of handling that event data and is the subject of the rest of this paper.

We will examine a series of tools to enhance this SIEM process. Software for advanced data analysis of event data flows, will be used, examining event data from systems configured to forward that data. From this implementation, we will utilize a customizable web front end to review data, and change our search paradigms on the fly. With such powerful tools, unusual data, events, and activity on our network can be pulled out and examined. This capability is brought about from the merger of several different software sets running on Linux systems, including *Elasticsearch* (a full-text, distributed search engine), and *Kibana* (data visualization software for Elasticsearch), working with *Syslog-ng* log server software to parse and forward the data. The specific methods used are discussed in Section 2.0. To summarize, this process is a real-time search on active event data flows than can be customized as desired.

Consider the example of a concerned system administrator worried that a hard drive may be failing, on some system connected to the network. At first, a search query – perhaps initially looking at all Syslog Emergency, Alert, and Critical severity messages, combined with the facilities of Kernel or Daemon types (those most associated with hardware errors of this kind) – is used as an initial query, and then a more detailed focus is implemented on specific hard drive messages, perhaps even manufacturer-specific messages, pulled from across the network. Indeed, with this software, such a search can be conducted in under a few minutes. A similar query can just as easily focus in on a series of bad logins (perhaps someone scanning the network), or some sort of activity that might be keyed to a specific set of malware.

2. Material and Methods

As promised, below is a simplified approach to deploy this setup that should be useful for a small network. The goal here is not to discuss a complicated SIEM framework for a large network or exhaustively review the minutiae of Kibana/Elasticsearch for specific search patterns; the goal is to give a stripped-down setup that can then be customized later for specific needs. This configuration gives powerful capabilities to an operations team to monitor their respective networks.

A diagram may be useful here:

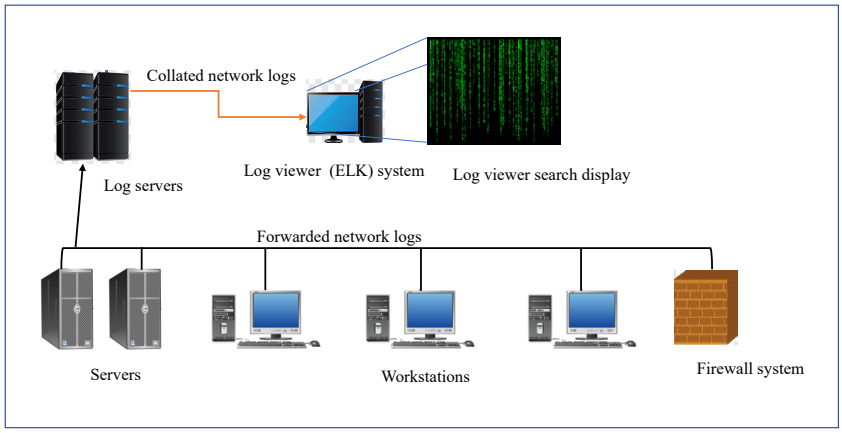

Fig. 1. Network log forwarding and collating for SIEM.

The architecture model is straightforward. Standard network logging is set up on workstations, servers and other systems in the network, that forward copies of the system event logs to a central log server (see Fig. 1). This log server, considering our discussions of security-fault tolerance in Section 1.1.1, should be exceptionally resilient and hardened against security threats – the one task of the server is to securely gather, store and forward these system logs. It should not run other applications or services. These collated logs are then forwarded over to a log viewer system containing the Elasticsearch, and Kibana toolsets where those logs are processed for immediate real-time review. (This is often called an ELK system for Elasticsearch, Logstash, and Kibana, however the setup described here does not utilize the Logstash libraries.) The log viewer has a much more complicated software setup and is much more difficult to secure – however in our design here these logs are *copies* of what is being stored elsewhere. Even in the event of a total system failure, a static backup of the log viewer (perhaps in a warm state) can quickly have the logs restored to it from log storage. What we show here makes the assumption the log viewer is not reliable, however the log viewer system can be made more reliable, with advanced backup and redundancy if so desired. To be clear, this configuration is designed to enable the essential capabilities necessary for real-time analysis – more complex configurations can rapidly grow to require enhanced security on the log viewing system, and for detailed long arc analysis techniques may require aggressive backup of the *state* of the log viewer.

These next sections cover the specific software, steps, and configurations used to set up this environment. As this uses open source, freely available software, this description is meant to be sufficient to set up an equivalent network-monitoring setup and conduct real-time analysis for a moderate-sized network.

2.1. Software Deployment and Version Practicum

This section makes several assumptions about the underlying base configuration of the local network. They are as follows:

1) System and other software logs have been set up to forward to a remote syslog-based log server (this is standard best practice which we will not detail here), and those logs are stored and backed up securely.*
2) The system state is reasonably close to a sterile implementation (i.e. results from the systems providing system and other security-event logs are valid and free of compromise).
3) Like the log server, hardware is available so that the log viewer can operate on a system dedicated to this purpose.
4) For the sake of simplicity of design (i.e. the KISS principle [14]) this setup seeks to be clear and avoid unnecessary complexity. For example: some later versions of some of the software sets use multiple layers of `ssl` to protect larger installations against external access and fold in complex password management schemas. This setup handily avoids the problem by not permitting any external access to the system – users must log in to the system using standard system security (such as `ssh`).

* The author recommends the use of a log server with massive overcapacity in both storage, memory, and performance, as well as at least one read-only archival backup system. For moderate-sized networks (in terms of a team-level configuration, *not* coverage of the whole organization) this should not be a significant investment in most cases.

2.1.1 Software Versions and Software Installation/Deployment

As the objective of this paper is to serve as a cookbook for assisting other deployments, discussion of software selection and installation is crucial. We will look at each of the components in the order of installation. On the log server, no software installation should be required. However, the log server should be able to forward log data to the log viewer system without any impact on its own performance or function (i.e. degradation in response time or, worse, dropped logs). For our own setup, we use a highly hardened OpenBSD system configured with minimal services and software – literally all it does is process and store network logs. With a few configuration changes, as well as careful examination of the performance impact over time, this was easily implemented. (How to determine what the correct configuration changes to these software sets is documented in the next subsection.)

For installation on the log viewer this was a more complex process. As noted, this involved the installation and configuration of several pieces of software in order. As software version control for open source projects is challenging (!), what is documented here will reflect the process of obtaining the most current stable version to replicate our efforts, rather than encourage hunting down specific versions of software that will be out of date before this paper is published (when specific versions are noted here, it is for the purpose of example). These software sets are installed on Red Hat Enterprise Linux systems using standard package management services, such as the `yum / dnf` package management systems, the `rpm` package manager commands, and conventional Linux / Unix software tools. This does also require access to the specific software repositories for the operating system being used (e.g. Red Hat and variants, or associated components in other Linux distributions). Also, one of the reasons for using `yum` / `dnf` and `rpm` is to ensure the automatic installation of software dependencies – understand that millage may vary considerably depending on the operating system setup, and diligence must still be exercised to check that the dependencies of each of the software sets are met. The use of the specific software repositories, and direct download of the respective `rpm` package files, will be demonstrated – use either method as desired. Either "`yum install" or` standard `rpm` commands can be used to install these packages.

As a preliminary step, the following software repositories can be added to the system repository list (nominally in `/etc/yum.repos.d/`) by commands of the form:

```
% cd /etc/yum.repos.d/
```

For Syslog-ng, the repository used for our system (update to latest build version from [15]) is:

```
% wget https://copr.fedorainfracloud.org/coprs/czanik/syslog-ng321/repo/epel-
7/czanik-syslog-ng321-epel-7.repo
```

For the Elasticsearch repository, create a file with the following content (change to suit your Red Hat version), and create and save it as follows [16]:

```
% vi /etc/yum.repos.d/elasticsearch.repo

[elasticsearch-7.x]
name=Elasticsearch repository for 7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
gpgcheck=1
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch
enabled=1
autorefresh=1
type=rpm-md
```

And install the associated GPG key, and enable the repository:

```
% rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
% yum --enable elasticsearch
```

The first component to install is the updated Syslog-ng software set, which offers content based, rich filtering capability and enhanced flexibility over other Syslog implementations. While Rsyslog, the standard syslog package on Red Hat and other distributions of Linux, offers many of these capabilities as well, one of the significant difficulties in implementation is the *correct configuration* of these software sets to work together. To avoid excess complication, we did not want to use different syslog software on the log server and log viewer systems – an update to this paper can

examine porting this capability to Rsyslog. Installation of Syslog-ng can be effected by a command of the form (note: `yum` is an alias in Enterprise 8 and later versions of Red Hat):

```
% yum install syslog-ng syslog-ng-java
```

Syslog-ng may also be directly downloaded by commands (be careful to ensure dependencies are met) of the form:

```
% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-
x86_64/00863414-syslog-ng/syslog-ng-3.20.1-1.el7.x86_64.rpm

% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-
x86_64/00863414-syslog-ng/syslog-ng-http-3.20.1-1.el7.x86_64.rpm

% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-
x86_64/00863414-syslog-ng/syslog-ng-java-3.20.1-1.el7.x86_64.rpm

% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-
x86_64/00863414-syslog-ng/syslog-ng-python-3.20.1-1.el7.x86_64.rpm

% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-
x86_64/00863414-syslog-ng/syslog-ng-riemann-3.20.1-1.el7.x86_64.rpm

% wget https://copr-be.cloud.fedoraproject.org/results/czanik/syslog-ng320/epel-7-x86_64/00863414-
syslog-ng/syslog-ng-smtp-3.20.1-1.el7.x86_64.rpm
```

This will install a (Red Hat) standard set of Syslog-ng packages with the necessary Java dependencies – used for integration with Elasticsearch.

The second component to be added to the system is the Elasticsearch search engine software set. Installation once the repository is set up and enabled is likewise:

```
% yum install elasticsearch
```

or direct download (again checking dependencies) of:

```
% wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.01.8-x86_64.rpm
```

The third component of this configuration is the Kibana data visualization software. With the added Elasticsearch repository, this software is pulled from the same source:

```
% yum install kibana
```

Or direct download (again checking dependencies) of [17]:

```
% wget https://artifacts.elastic.co/downloads/elasticsearch/kibana-7.0.1-x86_64.rpm
```

Along with the Elasticsearch and Kibana toolsets, it is recommended that an optional web administration tool named Cerebro should be added to the system to provide additional administrative support for these Elasticsearch instances. This software can be downloaded as follows from GitHub to a local directory and installed [18]:

```
% wget https://github.com/lmenezes/cerebro/releases/download/v0.8.5/cerebro-0.8.5-1.noarch.rpm
```

2.2. Software and System Configuration

The configuration implementation details provided below were gathered from the author's implementation efforts, and involved some trial and error to achieve a working setup.

2.2.1 Log Server Configuration

The first step of implementation is to prepare the log server to forward log data to the external log viewer and to have the log viewer prepare to receive this data and hand it over to the Elasticsearch search engine software. While working through the most optimal settings for both network and system performance takes some trial and error, the actual configuration modifications are not large. In the Syslog-ng configuration file (syslog-ng.conf) on the log server, the following two updates were made:

1) In the front part of the file optimization was undertaken to increase buffers and improve efficiency to better handle incoming TCP and UDP log events:

```
options {
        use_dns(no);
        use_fqdn(no);
        create_dirs(no);
        keep_hostname(yes);
        chain_hostnames(off);
        flush-lines(200);
        flush-timeout(5000);
# timeout is in msec
};

# Input section
source s_net {
        network ( transport("udp") port(514) log-iw-size(25000) max-connections(150) );
        network ( transport("tcp") port(514) log-iw-size(25000) max-connections(150) );
        network ( transport("tcp") port(989) );
};
```

2) This later section sets up the log transfer to the log review server. Note that the User Datagram Protocol (UDP) protocol was found to handle higher log event loads while failing more gracefully (*fail-gently*). Some details have been obscured.

```
###
#
# ELK/LS transfer section – for forward to review server.
#
###

destination ELK {

        network(
                "132.xxx.xxx.xxx"
                transport("udp")
                port(xxxx)
                log-fifo-size(10000)

        );

};

###

#
# logs all incoming messages from network source to the host sorter
#

log { source(s_net); destination(Hosts);  destination(ELK); };
```

These configuration changes imposed little additional system load and helpfully required no additional software on our tightly configuration-managed and hardened log server. As a one-way data transfer (see Fig. 1.), this helpfully does not entail any new security concerns for the log server besides the release of the log events.

2.2.2 Log viewer configuration

The configuration of the multiple software sets on the SIEM log viewer system is necessarily more complex. We begin by configuring the log viewer to accept the incoming security event logs from the log server, which we set up in the previous section. The setup is similar to the one on the log server being the other side of this network "pipe":

```
options {
    flush_lines (200);
#    flush-timeout(5000);
    time_reopen (10);
    log_fifo_size (1000);
```

```
    chain_hostnames (off);
    use_dns (no);
    use_fqdn (no);
    use-uniqid(yes);
    create_dirs (no);
    keep_hostname (yes);
};
```

(Note the commented out Transmission Control Protocol (TCP) section here – as discussed, for our situation, UDP was the more reliable transport.)

```
# Input/Output section
source s_net {
     network ( transport("udp") port(514) log-iw-size(25000) max-connections(150) );
#    network ( transport("tcp") port(514) log-iw-size(25000) max-connections(150) );
     network ( transport("udp") port(3454) log-iw-size(25000) max-connections(150) );
#    network ( transport("tcp") port(3454) log-iw-size(25000) max-connections(150) );
};
```

This latter section on the log viewer Syslog-ng configuration takes in the incoming event logs and collates them to send to the Elasticsearch engine. It is comprised of two parts – a set of destination configurations, and a tag that links the source and destination locations:

```
#
destination EHosts {
    elasticsearch2(
            client-lib-dir("/usr/share/elasticsearch/lib/:/usr/lib/jvm/java-1.8.0-openjdk-
        1.8.0.191.b12-1.el7_6.x86_64/jre/lib/amd64/server/:/usr/lib64/syslog-ng/java-modules/")
            index("syslog-ng")
            type("syslog")
            resource("/etc/elasticsearch/elasticsearch.yml")
            client_mode("http")
            flush_limit("0")
    );
};
#

#

log { source(s_net); destination(EHosts); flags(flow-control); };
```

For testing purposes for the Syslog-ng service, once the changes on both the log server and the log viewer client are implemented and the respective services are restarted, the use of the "`syslog-ng-ctl stats`" command will prove useful to ensure the forwarding and processing of logs. Values of

```
…;dropped; [number of logs]
…;processed; [number of logs]
…;queued; [number of logs]
…;written; [number of logs]
```

for the queues will give indication of how the event log data is flowing through the connection and, more importantly, where it may have stopped.

With this set up, we now turn towards the configuration of the Elasticsearch engine. One of Elasticsearch's strengths is its automatic provisioning and re-optimization of server resources due to varying demand on the system, so little configuration is necessary. Although Elasticsearch will adjust its settings based on the number of CPU cores and filesystem space, it does not adjust for memory usage, and it is a strongly desired in a configuration to keep the Elasticsearch Java engine in memory and prevent it from swapping out (in fact it may be desirable to disable swapping entirely in production). The minimum and maximum heap memory sizes should be set to approximately 50% of the total system memory size, and the two heap memory size limits may be configured in the configuration file `/etc/elasticsearch/jvm.options` as follows:

```
################################################################
## IMPORTANT: JVM heap size
################################################################
##
## You should always set the min and max JVM heap size to the
## same value. For example, to set the heap to 4 GB, set:
##
## -Xms4g
## -Xmx4g
##
## See https://www.elastic.co/guide/en/elasticsearch/reference/current/heap-size.html
## for more information
################################################################
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
-Xms2100m
-Xmx2100m
```

Once this is accomplished, only a few further configuration customizations need be done. For our setup we configure the system as a hot image of the SEC392 cluster. Particular node name(s) are given while various directory configurations are set up for our particular filesystem. Finally, we set up access on the localhost address on port 9200. These settings can be found in the `/etc/elasticsearch/elasticsearch.yml` configuration file like so (note that some details have been obscured):

```
# Use a descriptive name for your cluster:
#
cluster.name: SECxx
#

# Use a descriptive name for the node:
#
node.name: NCCxx

# Add custom attributes to the node:
#
node.attr.box_type: hot
path.repo: ["/users/backups"]

#
# Path to directory where to store the data (separate multiple locations by comma):
#
path.data: /users/elasticsearch
#
# Path to log files:
#
path.logs: /var/log/elasticsearch

# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: 127.0.0.1
#
# Set a custom port for HTTP:
#
http.port: 9200
#
```

Once the Elasticsearch engine is started, for testing one can connect up to localhost at port 9200 with a web browser, where a JSON response will be returned of a form similar to Fig. 2.

Fig. 2. Test response from Elasticsearch API on port 9200.

Note that the JSON response will vary depending on your particular settings.

With Elasticsearch running, we now proceed to the setup of the Kibana data visualization tool and the Cerebro administration set. As Kibana is integrated with Elasticsearch its installation is a simple one and no modifications in the configuration are expected. However, Kibana will be used to extensively customize the sets of data examined and the visualizations of that data. Likewise, Cerebro by default does not require customization (although it does require access to the Elasticsearch API on port 9200), but offers considerable capability for customization, backup and administration of Elasticsearch.

The setup of the log instance (index) in the Cerebro tool is briefly covered here. (This configuration, like the rest of our discussion is described in a clear and plain manner as an example and is not production optimal for larger sites. It also does not include replicas or backup in its configuration and again has a simplified access policy. Production systems should use the redundant and backup capabilities offered by these toolsets and can spread the load across multiple machines for further improved fault tolerance.)

Login to the Cerebro tool via a web browser (nominally running at `http://localhost:9000`). Upon startup, a cluster display similar to that shown in Fig. 3 will be obtained showing the node(s) and cluster(s) being managed (note that some details are obscured in the figure). After selecting the *create index* option under the *more* menu, an index is generated to receive logs, as shown in Fig. 4.

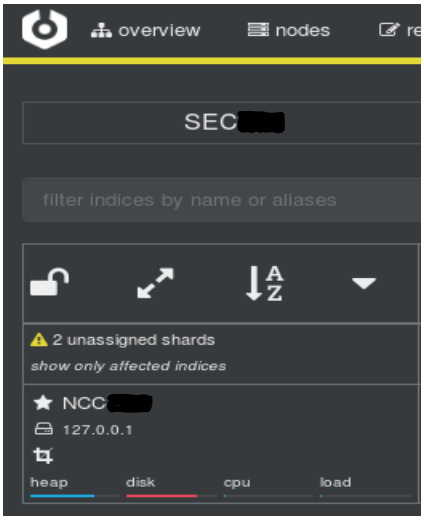

Fig. 3. Initial Cerebro login.

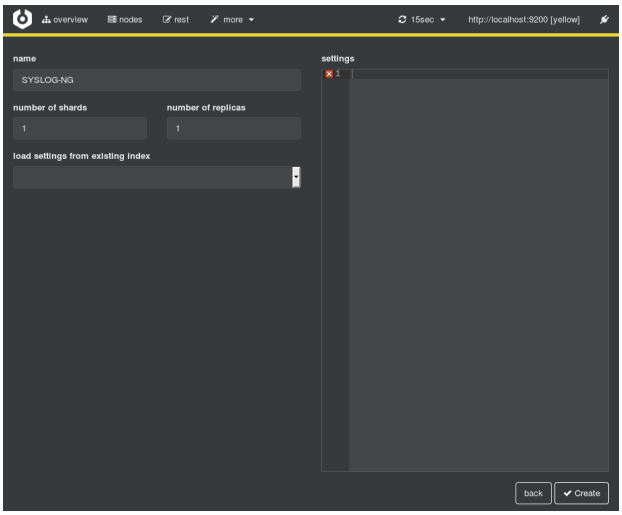

Fig. 4. New index generation page.

When the index generation is complete the resulting index (Fig. 5.) can be observed on the cluster, containing one primary and one replica Elasticsearch shard (a fundamental Elasticsearch data unit). Note that this index is empty of data (containing no docs). It must be referenced in the Syslog-ng configuration file and the configuration reloaded to begin receiving data. Further customization and backup configuration can be undertaken as needed.

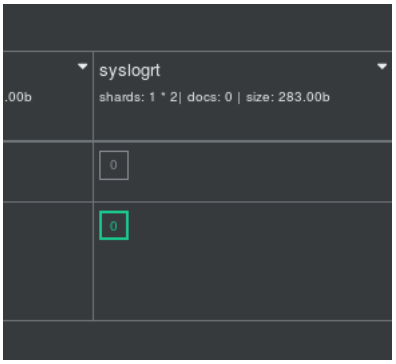

Fig. 5. Resultant index.

With this configuration, we can now browse to the Kibana service (nominally running at `http://localhost:5601`). If Kibana is running properly a default screen similar to the one shown in Fig. 6 will be displayed, showcasing much of the capability of the Kibana toolset.

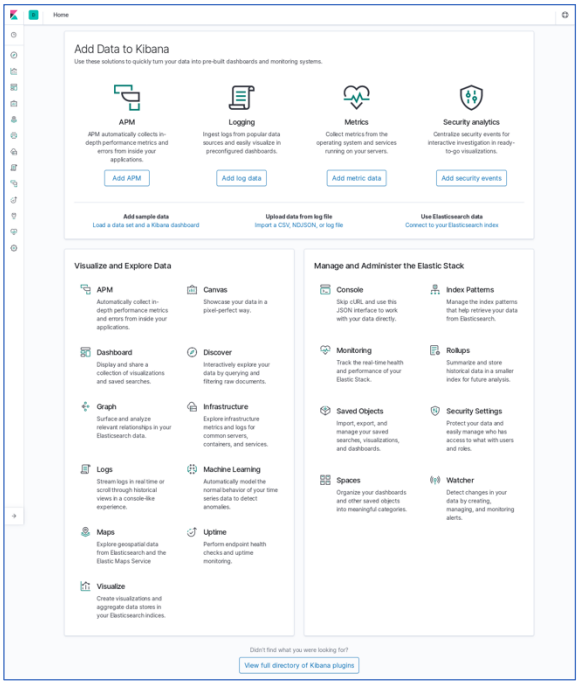

Fig. 6. Kibana default login screen.

While this listing is helpful, of greater use is a test that confirms the installed software components and systems are actually transferring data to the Elasticsearch index, and that Kibana can manipulate and display them. In the *Management* side tab fill in the index pattern with the name of the index created in Cerebro and referenced in the Syslog-ng configuration file. The resultant data (once logs are flowing) should look similar to that shown in Fig. 7.

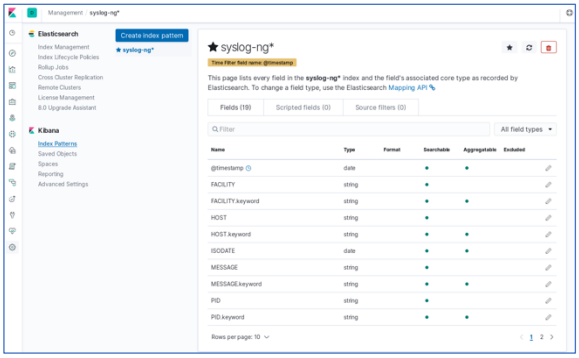

Fig. 7. Fields in received syslog event logging data.

Even more useful as a test case is an actual chart of the incoming data. Select the *Discover* side tab, and then, making sure the index selection has not been changed (syslog-ng*, as seen in Fig. 7), select a date range in the upper right-hand corner. (*Last 30 minutes* is a good selection here.) A histogram similar to that shown in Fig. 8 will be generated.

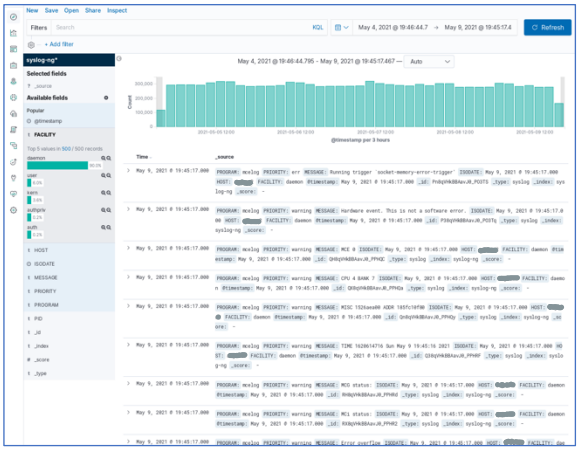

Fig. 8. Histogram over time of security event data.

Of even greater interest is the ease of customization of this interface. By selecting any of the dynamic characteristics on this page the display can be focused on those details. By selecting one of the options under Facility such as “user,” a filter for just “user” Facility events will be selected, or one of the options under Priority, such as “Warning” only “Warning” Priority events will be selected. Filters can be combined such as “user” Facility and “Warning” Priority for a more granular examination of data. Specific data patterns and values can be searched for. Furthermore, graphs of this data counting events on the y- and x-axis can be created by selecting the *Visualize* sidebar tab, then a standard *Line plot*, enabling one to very rapidly generate a line chart of events happening over time (Fig. 9.).

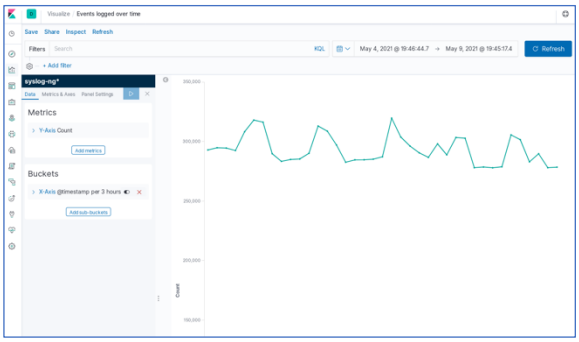

Fig. 9. Plot of security events over time.

With these last two displays, it is clear that the SIEM components are not only functional but also producing valuable data. These insights into the state of the network can greatly decrease the time to detect (Dt) security events. Further visualization examples will be considered in the next section.

3. Calculation and Results

Consider some more advanced visualizations and customizations that can be applied easily to the displays and data. Much of this effort is made dramatically simple by the efficiency of the Kibana tool and its structure that allows near-infinite review and modification of its searches. Again, this is not meant to be a definitive text but merely a proof of concept of the software sets’ capability – other resources can lead to more advanced, granular examinations. As discussed in Section 4, other data sets can easily be incorporated as well.

One of the more useful parts of the Kibana architecture is the approach to modularity in its search mechanism and the ability to layer those searches in increasingly complex ways. We will look as some simple searches and then see how they can be incorporated into multilayered displays that give a bird’s-eye view of the whole network. In order to keep this discussion as clear as possible, this paper will incorporate basic searches, using the particular configuration terms from the interface, and present the charts of the data as specific examples.

The first search we look at is a simple one. “What hosts on the network are having the most problems?” Going back to our first example (in Section 1.3), this could be a system with a failing hard drive. While it may not be an

obvious problem, the top hosts in that list clearly have something amiss. It may be a hardware problem, a software misconfiguration, or perhaps something worse. In any event, it is good to check on the system in question.
This chart is easily generated. Under the *Visualize* tab, select a vertical bar chart of the index, with a y-axis of *Count* and a x-axis of *Bucket* (group), using *Aggregation of Terms*, from the *HOST keyword*, under *Order By Count* (see Fig. 10).

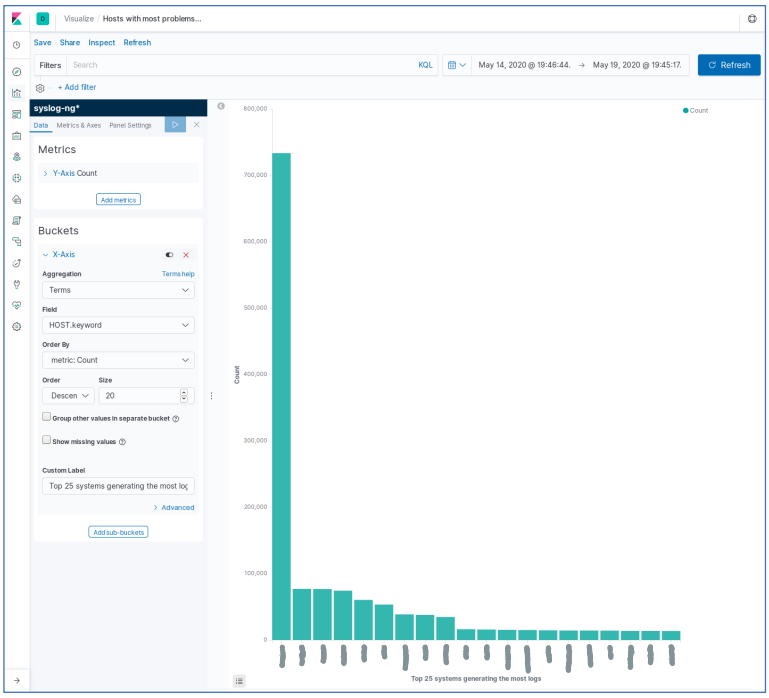

Fig. 10. Hosts with the most problems.

What makes Kibana so impressive is that Fig. 10 can be constructed to summarize results in under a minute. With a little bit of cleverness, logical filter functions can be added to allow one to zoom in on systems that are particularly troubled. This is accomplished by adding a filter function to the previous display so that it only displays the most severe errors (by *not* displaying: Notice, Info, Debug, Warning, or Err priorities – e.g. Emerg, Alert, and Crit), as seen in Fig. 11.

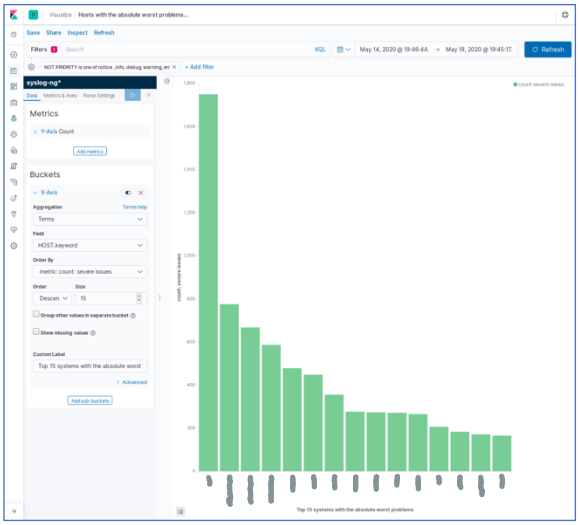

Fig. 11. Hosts with the absolute worst problems.

It will be left up to the reader to consider how one might make a "top-two" syslog priority level chart (Emerg and Alert) – perhaps titled *Is the CPU on fire?* Not a security specific display, but one which should attract the attention of a system administrator in any event.

Another type of chart that showcases Kibana's capabilities is a pie chart, particularly one that combines two (or more) sets of data, with a filter function as above. Consider a pie chart, using *Count* for slice size and *Aggregation of Terms* (for both split slices), one using *PRIORITY keyword Count*, and the other *HOST keyword Count*. Adding in a filter for high-priority issues, a useful display is produced (see Fig. 12.) that has several interesting attributes.

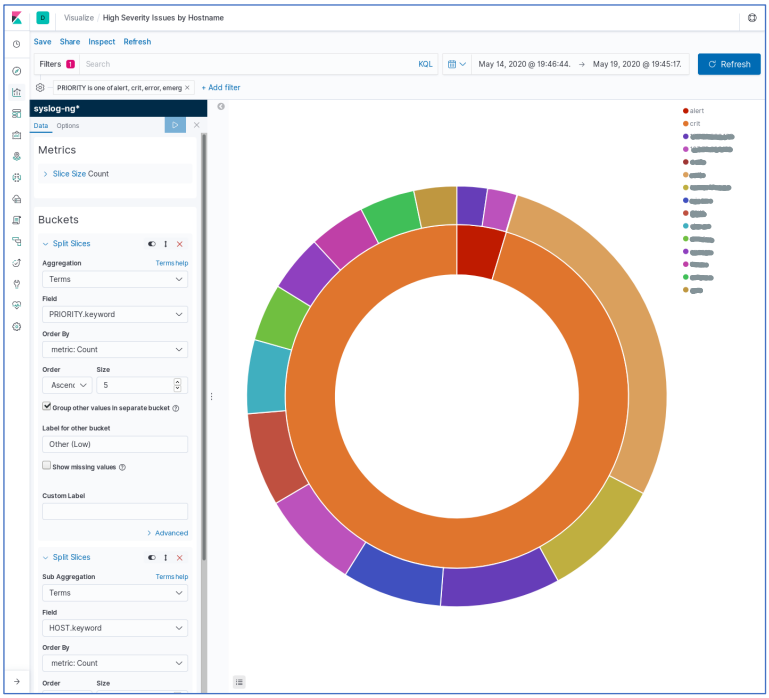

Fig. 12. High-severity issues by hostname

Priority-level issues are grouped with the associated hostnames, so that issues with networked systems can be quickly sorted into the most critical concerns – and the associated hostnames. Which machine has the most severe, and the greatest number of problems? Moreover, by hovering the pointer over the area of the chart desired, more detailed statistics are presented. Also, by clicking on a particular system or priority, a configuration dialog appears that allows additional filters to be applied to the chart (see Fig. 13.).

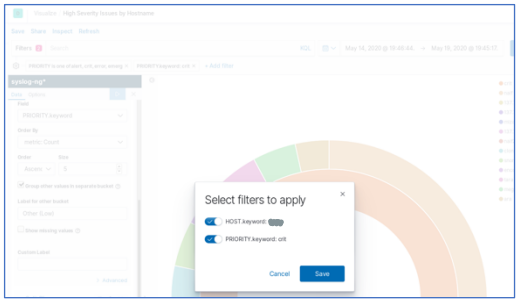

Fig. 13. Additional filters dialog.

This has been even expanded to generate a display of three data sets: syslog priority, host, and facility for further capability. However, even more useful is the fact that these displays may be merged together, providing yet more comprehensive data about the underlying network. Under the *Dashboard* sidebar tab, a capability to incorporate multiple linked visualizations in a single display is available. By simply selecting the *Add* option, previously created visualization displays can be added, moved and resized on a single dashboard. With the ability to drill down to particular hosts, problems, or priorities as described in Fig. 13, the power to gather and customize the data display is a

potent one. A couple examples of this capability are examined here. Consider the combination of an “Events logged over time” chart with a “Severity by Hostname” chart, as well as a log listing over time (which becomes particularly useful as filters are applied to zoom in on specific cases), as seen in Fig. 14.

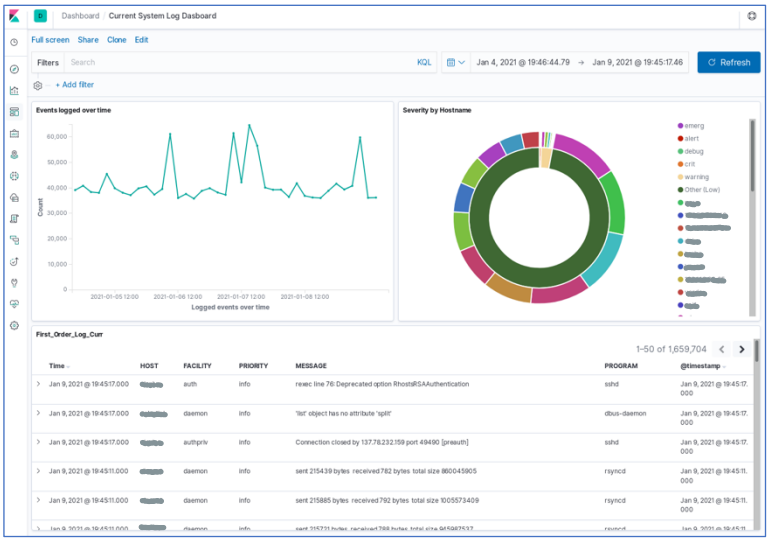

Fig. 14. System Log View Dashboard

With this combination of visualizations, and the ability to zoom in and apply arbitrary filters to the data, network problems and system events become easy to discern. With specific filters applied for particular event tags, much more rapid analysis and response to problems are possible.

As a final demonstration, we finish this examination with an example of a “kitchen-sink” display that covers a wide array of potential real-time display approaches (Fig. 15).

Fig. 15. System Log Dashboard Super Edition.

4. Discussion and Future Efforts

The purpose of this paper is to provide a proof of concept for a modest SIEM architecture appropriate for a moderately sized ground data system or similar environment. Such a system is not a substitute or panacea for a thorough and well-established security architecture, well trained systems administrators, and a dedication to system review and improvement. Well used, however, it is an effective part of the whole. The ability to quickly detect unwanted activity or failure on a network can improve its reliability and security posture.

Hopefully the examples and descriptions here are sufficient for the reader to replicate our efforts on their own network. It is a significant challenge to determine the correct level of detail for such a text, and especially in open source projects where version control and backwards compatibility can be problematic, without overwhelming the reader with stupefying detail. More examples of search approaches could have been presented here, but effort was taken to present a summary of basic capabilities that others could build upon. Likewise, every effort has been taken to demonstrate the mechanism used in this effort; however, the reader should be aware several of the settings, particularly with Syslog-ng are very fussy in syntax and structure – the configuration documented here was derived from less-than-optimal documentation after some trial and error. Furthermore, as has been noted, this setup is a proof of concept, i.e.

a prototype, and is not fully configured with the backup, redundancy and security measures needed in a production environment. Most of those measures, once a working setup is available, are fairly straightforward to implement.

Looking forward to future improvements (above and beyond enterprise-level bulletproofing of the environment), adding more event logging and data types to the Elasticsearch engine from other systems and software can give even more detail about the functioning of our ground data system. The use of Logstash or other software to forward additional data from key machines and services, especially file system and process accounting logs, would be invaluable. The advantage of Elasticsearch/Kibana is the ability it provides to rapidly search and display data, and the more data we deploy to the SIEM, the more useful that data becomes.

Ultimately, the author's interest in this setup stems from many years of trying to implement – badly – some type of data visualization tool or system. These tools have capabilities that would have been difficult to imagine years ago, but are invaluable today. With so much in the computer security environment offering little positive encouragement these days, the Elasticsearch architecture provides real assistance for security and operations.

Acknowledgements

To the Systems Engineers and System Administrators who have supported the security architecture discussed here I would like to give my sincere thanks. Additionally, of the editors who assisted with this paper, Dr. Zachary Porcu and Kristine McGowan did tremendous service and to both of them a considerable debt of gratitude is owed.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference to any specific commercial product, process, or service by trade name, trademark, manufacturer or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology. © 2023 California Institute of Technology. Government sponsorship acknowledged.

References

[1] R. Beswick, P. Antreasian, S. Gillam, Y. H. Hahn, D. Roth, J. B. Jones, Navigation Ground Data System Engineering for the Cassini/Huygens Mission, AIAA 2008-3247, SpaceOps 2008 Conference, Heidelberg, Germany, 2008, 12–16 May. doi: 10.2514/6.2008-3247

[2] R. M. Beswick, The Cassini/Huygens Navigation Ground Data System: Design, Implementation, and Operations, in: Helene Pasquier, et al. (Eds.), Space Operations: Inspiring Humankind's Future, Springer Nature Switzerland AG, 2019, pp. 261-322. doi: 10.1007/978-3-030-11536-4

[3] R. M. Beswick, D. C. Roth, A Gilded Cage: Cassini/Huygens Navigation Ground Data System Engineering for Security, AIAA 2012- 1267202, SpaceOps 2012 Conference, Stockholm, Sweden, 2012, 11–15 June. doi:10.2514/6.2012-1267202

[4] R. M. Beswick, Computer Security as an Engineering Practice: A System Engineering Discussion, IEEE: 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT), 2017, 27–29 September. doi: 10.1109/SMC-IT.2017.18

[5] R. M. Beswick, Computer Security as an Engineering Practice: A System Engineering Discussion, Advances in Science, Technology, and Engineering Systems Journal, 4 (2019) 357–369. doi: 10.25046/aj040245

[6] J. Henderson, J. Hubbard, Security 455: SIEM Design and Implementation, SANS Cyber Defense Initiative Conference, Washington, D.C., 2018, 11–18 December.

[7] B. Johnson, Fault-Tolerant Microprocessor-Based Systems, IEEE Micro, 4 (1984) 6–21.

[8] J. Gray, D. P. Siewiorek, High-Availability Computer Systems, IEEE Computer, 24 (1991) 39–48. doi: 10.1109/2.84898

[9] R. Anderson, R. Needham, Programming Satan's Computer, in: van Leeuwen, J. (Eds.), Computer Science Today, Lecture Notes in Computer Science, 1000, Springer, Berlin, Heiderlberg, 1995, pp. 426–441.

[10] O. S. Saydjari, Engineering Trustworthy Systems, McGraw-Hill Education, New York, 2018.

[11] M. Bishop, Computer Security, Art and Science, 2nd. Ed., Addison-Wesley, New York, 2019.

[12] W. Schwartau, Analogue Network Security, Schwartau Haus, 2020.

[13] W. Schwartau, Time Based Security, Interpact Press, Seminole, Florida, 1999.

[14] B. R. Rich, Clarence Leonard (Kelly) Johnson 1910-1990, Biographical Memoirs, Vol. 67, National Academies Press, Washington, D.C., 1995, p. 231.

[15] P. Czanik, Installing latest syslog-ng on RHEL and other RPM distributions, 5 March 2021, https://www.syslog-ng.com/community/b/blog/posts/installing-latest-syslog-ng-on-rhel-and-other-rpm-distributions, (accessed 02.01.2023).

[16] Installing Elasticsearch, 2023, https://www.elastic.co/guide/en/elasticsearch/reference/current/install-elasticsearch.html, (accessed 02.01.2023).

[17] Installing Kibana, 2023, https://www.elastic.co/guide/en/kibana/current/install.html, (accessed 02.01.2023).

[18] Cerebro, 10 April 2021, https://github.com/lmenezes/cerebro, (accessed 02.01.2023).