rt-solutions.de
experts you can trust.

Open-Source – SIEM

Daniel Mahrenholz, Ralf Schumann

rt-solutions.de GmbH
{mahrenholz, schumann}@rt-solutions.de

Zusammenfassung

Die Protokollierung sicherheitsrelevanter Vorgänge sowie deren Auswertung sind ein wichtiges Element anerkannter Sicherheitsstandards und Best Practices. Praktisch werden diese Maßnahmen aber durch viele Unternehmen und Organisationen nicht umgesetzt. Häufige Gründe für den Verzicht sind die Komplexität sowie Kosten kommerzieller Lösungen. Oft existieren auch spezielle Anforderungen besonders im Bereich der IT-Forensik bzw. des Datenschutzes, die die existierenden kommerziellen wie Open-Source-Lösungen nicht erfüllen. Die Realisierung eines Systems auf Basis von Open-Source-Komponenten im Eigenbau kann hier eine effektive Alternative bieten. Die notwendigen Bausteine sowie Erfahrungen aus Implementierungsprojekten sollen in diesem Artikel näher betrachtet werden.

1 Einleitung

Die zentralisierte und strukturierte Protokollierung sicherheitsrelevanter Ereignisse, des Betriebszustandes sowie von Störungen oder Ausfällen von IT-Systemen bildet einen wesentlichen Bestandteil anerkannter Standards wie dem BSI-Grundschutz oder der ISO27002 sowie Best Practices. Untrennbar ist dies mit der Notwendigkeit verbunden, die protokollierten Informationen systematisch auszuwerten und auf Betriebsstörungen oder Sicherheitsvorfälle zu reagieren. In Summe bieten diese Maßnahmen eine Möglichkeit, die Verletzung von Integrität, Verfügbarkeit und Vertraulichkeit der IT-Systeme und verarbeiteten Informationen zu erkennen, zu analysieren und zeitnah zu behandeln, um diese nachhaltig zu verbessern. Über die Ausgestaltung der Maßnahmen, d.h. welche Informationen wie protokolliert und nach welchen Kriterien ausgewertet werden, treffen die Standards und Best Practices oft nur allgemeine Aussagen, denn dies ist maßgeblich von der Bedrohungslage und Risikobewertung sowie der technischen Infrastruktur eines Unternehmens abhängig. So sind sich die Standards und Best Practices z.B. einig, dass die Nutzung von privilegierten Rechten („Sonderzugriffsrechte“) aufgezeichnet und ausgewertet werden sollte und dabei Informationen wie Zeitstempel, betroffenes System, aktiver Benutzer und durchgeführte Aktionen erfasst werden müssen, es werden aber keine Vorgaben gemacht, wie dies im Detail erfolgen soll. Durch die Vielzahl möglicher Systeme und Anwendungen sowie Bedrohungen ergeben sich vielfältige Anforderungen, die ein existierendes System unmöglich Out-of-the-Box komplett abdecken kann. Oft besteht deshalb die Möglichkeit, ein existierendes System (kommerziell wie Open Source) anzupassen oder zu erweitern. Daneben besteht aber auch die Möglichkeit, ein System komplett selber zu entwickeln. Da diese Option mit vielen Unbekannten besetzt ist, soll in diesem Artikel, ungeachtet der anderen Optionen, beschrieben werden, was für die Realisierung im Eigenbau notwendig ist. Dabei sollen auch die Erfahrungswerte und Schwierigkeiten beschrieben werden, um ein derartiges Projekt besser einschätzen zu können. Die im Einzelfall effizienteste Lösung sollte vorab aber immer in einer Kosten-Nutzen-Analyse betrachtet werden.

2 Begriffsdefinitionen

In diesem Artikel werden die Begriffe „Log Management" (LM) und „Security Information and Event Management" (SIEM) in der folgenden Bedeutung verwendet.

Unter LM wird ein System verstanden, das zentralisiert Loginformationen von unterschiedlichen Quellen aufnimmt, diese normalisiert, d.h. einheitliche Datenfelder aus unstrukturierten Informationen extrahiert, speichert und für die Suche und das Reporting verfügbar macht. Derartige Funktionalitäten werden oft auch als „Security Information Management" (SIM) bezeichnet, dieser Begriff aber zur Vermeidung von Verwechslungen nicht verwendet.

Unter einem SIEM-System wird ein System verstanden, dass zusätzlich zu den Funktionalitäten des LM die Möglichkeit bietet, sicherheitsrelevante Ereignisse aus unterschiedlichsten Quellen in nahezu Echtzeit zu korrelieren, Muster zu erkennen und durch Alarmierungen zu melden, um hierdurch ggf. externe Aktivitäten oder Workflows auszulösen.

3 Basiskomponenten eines LM-/SIEM-Systems

Ein SIEM-System besteht aus den folgenden wesentlichen Komponenten:

- Datenerhebung
- Datenübertragung
- Log-Verarbeitung
- Langzeitspeicherung / Archivierung
- Suche und Reporting
- Korrelation und Alarmierung

Bei der Datenerhebung werden die Log-Informationen an der Quelle erhoben. Oft erfolgt dies durch die Quelle (z.B. das Betriebssystem oder die Anwendung) selber, ggf. auch durch zusätzliche Agenten, die auf der Quelle installiert werden.

Die Datenübertragung umfasst sowohl die Übertragung von der Quelle zum SIEM-System als auch zwischen den SIEM-Komponenten und zu den Nutzern des SIEM-Systems. Auch Proxy-Systeme, die Log-Informationen zwischenspeichern und weiterleiten, werden als Teil der Datenübertragung betrachtet.

Die Log-Verarbeitung beinhaltet Funktionen wie die Normalisierung, d.h. die Extraktion von einheitlichen Datenfeldern aus den typischerweise unstrukturierten Log-Informationen, die Klassifizierung und Kategorisierung, d.h. die Zuweisung einer verallgemeinerten und ggf. maschinenlesbaren Bedeutung des Log-Inhaltes. Enthalten sein können auch weitere Funktionen z.B. zum Umkodierung von Binärformaten, zur Integritätssicherung oder zur Verschleierung sensibler oder personenbezogener Informationen. Die Log-Verarbeitung wird in unterschiedlichen Systemen oft sehr unterschiedlich implementiert.

Die Langzeitspeicherung hat die Aufgabe die originalen Log-Informationen sowie die bei der Verarbeitung hinzugefügten Informationen persistent zu speichern. Sie muss dabei sowohl mit sehr hohen Datenraten als auch langen Aufbewahrungsfristen umgehen können. Typischerweise werden hier auch Funktionen wie die Integritätssicherung (z.B. kryptografische Checksummen gegen Verfälschung), Kompression sowie die Durchsetzung von Löschfristen implementiert.

Zur Analyse der aufgezeichneten Log-Informationen werden effiziente Such- und Reporting-Funktionen benötigt, die in der Lage sind, mit den sehr umfangreichen Archiven umgehen zu können und die Suchergebnisse derart zu aggregieren und zu visualisieren, dass sie für einen menschlichen Security-Analysten handhabbar werden.

Für die Realisierung eines SIEM-Systems wird zusätzlich eine Komponente zur Korrelation von normalisierten Log-Informationen benötigt, die inhaltliche und zeitliche Muster in nahezu Echtzeit analysieren und Treffer als Alarme an externe Komponenten melden kann.

4 SIEM-System im Eigenbau

Im Folgenden sollen die Entscheidungen und Erfahrungen beim Bau eines eigenen SIEM-Systems ausgeführt werden. Wie in der Einleitung beschrieben, können besondere Anforderungen eine solche Entwicklung rechtfertigen. Folgende besondere Anforderungen, die die Entscheidung maßgeblich beeinflusst haben, sollen deshalb vorab im Detail betrachtet werden:

- Besondere Forensik- und Analysefunktionen
- Datenschutzfunktionen
- Batch-Import von Log-Dateien und historische Korrelation

4.1 Besondere Forensik- und Analysefunktionen

Zur Planung eines LM-/SIEM-Systems ist es sehr wichtig, sich vorab intensiv damit zu beschäftigen, welche Informationen im System verarbeitet werden sollen und welche Suchen und Analysen darauf ausgeführt werden sollen.

Systeme zur Speicherung von Protokollinformationen haben lange Zeit relationale Datenbanken als Datenspeicher sowie zur Realisierung von Such- und Reporting-Funktionen genutzt. Durch das Wachstum der Logdatenvolumen sowie die Tatsache, dass es sich primär um unstrukturierte Daten handelt, wurden diese in den verschiedenen Lösungen in den letzten Jahren mehrheitlich durch sogenannte NoSQL-Datenbanken, d.h. Datenbanksysteme ohne zwingend festgelegte Tabellenschemata ersetzt. Hierdurch ist typischerweise auch eine wesentliche Analysemöglichkeit verschwunden – die Möglichkeit mehrere Ergebnismengen über eine JOIN-Operation zu verbinden. Dies ist z.B. notwendig, wenn ermittelt werden soll, welcher Nutzer auf eine externe IP zugegriffen hat, diese Information aber die Kombination zweier Informationsquellen erfordert (Log der Firewall und Log des Source-Systems mit den zu einem Zeitpunkt angemeldeten Benutzern). Verschiedene Systeme haben hierfür unterschiedliche Mechanismen implementiert:

1. Anreicherung von Events: Hierbei werden Events vor der Speicherung bereits um Kontextwissen zu einzelnen Datenfeldern (z.B. MAC-Adresse zu einer dynamisch vergebenen IP-Adresse) erweitert.
2. Sequenzen von Events: Zusammengehörige Events werden durch Suche vorgegebener Muster auf Basis von inhaltlichen und temporalen Gemeinsamkeiten gesucht und dem Nutzer zur Analyse überlassen (z.B. Anmeldung an einem System gefolgt von Zugriff auf eine externe IP-Adresse von diesem System).
3. Nachverarbeitung von Suchergebnissen – fehlende Analysefunktionen wie ein JOIN werden außerhalb der Datenbank implementiert.

Die Anreicherung von Events ist eine Standardfunktionalität in kommerziellen wie Open-Source-LM-Systemen. Der wesentliche Nachteil dieses Ansatzes ist, dass die verschiedenen Anreicherungen und die dafür notwendigen Informationen vorab bekannt sein müssen. Der wesentliche Vorteil hingegen ist, dass bei volatilen Informationen der zum jeweiligen Zeitpunkt der Anreicherung aktuelle Wert genutzt und somit persistent gemacht wird, ohne ihn zeitlich historisiert speichern zu müssen. Die Methode ist deshalb besonders geeignet für Informationen die zum jeweiligen Zeitpunkt bereits zweifelsfrei feststehen (z.B. dynamische Adresszuweisungen per DHCP). Für Informationen, die sich erst im Nachhinein ergeben (z.B. Reputation einer Internet-IP) ist es ungeeignet.

Die Erkennung von Sequenzen ist eine Kernfunktionalität von SIEM-Systemen, wird aber typischerweise nicht als Ersatz für einen JOIN verwendet, da auch hier die Muster vorab bekannt sein müssen und das Verfahren zudem oft eine Heuristik mit entsprechendem Fehlerpotential darstellt. Ein typisches Beispiel, dass die Grenzen aufzeigt, ist das Zuordnen von natürlichen Benutzern zu funktionalen Accounts (z.B. root auf Linux-Systemen). Meldet sich ein Benutzer mit einem personalisierten Account an und wechselt dann zu einem funktionalen Account, können die nachfolgenden Aktionen unter dem funktionalen Account durch den zeitlichen Zusammenhang einer Person zugeordnet werden. Melden sich aber mehrere Personen gleichzeitig an und verwenden den funktionalen Account parallel, ist eine Zuordnung nicht mehr zweifelsfrei möglich.

Die Option der Nachverarbeitung wird selbst bei kommerziellen Systemen selten genutzt. Splunk ist als einziges kommerzielles System auf eine Nachverarbeitung explizit ausgelegt [Splu16] und bietet einen (LEFT OUTER) JOIN zwischen zwei Suchanfragen an. Andere verbreitete, kommerzielle Systeme wie HP ArcSight oder IBM QRadar bieten nur einfache Lookup-Funktionen gegen vorbereitete Tabellen an.

Da das entworfene SIEM-System besonders intensiv für Forensik-Aufgaben genutzt werden soll, kommt es häufig vor, dass die Ergebnisse mehrerer Suchanfragen verbunden werden müssen, die vorab aber nicht vollständig bekannt sind, d.h. auf eine JOIN-Funktionalität kann deshalb nicht verzichtet werden. Auslöser für forensische Untersuchungen sind dabei von externen Stelle übermittelte Informationen zu Bedrohungen (Threat Intelligence) bzw. Anzeichen für erfolgreiche Angriffe (Indicators of Compromise, IOC) oder durch das SIEM-Regelwerk erkannte Abweichungen von der Norm bzw. von definierten Verhaltensstandards. Bei der Implementierung wurde wegen der jeweiligen Vor- und Nachteile auf eine Kombination aus Anreichung (z.B. für topologische Kontextinformationen oder GeoIP-Informationen) und Nachverarbeitung gesetzt.

4.2 Datenschutzfunktionen

Jedes LM-/SIEM-System berührt immer Fragen der Verarbeitung personenbezogener Informationen wie z.B. Benutzerkennungen, Email-Adressen oder auch IP-Adressen [DuKr14, DuKr09]. Zur Einhaltung der gesetzlichen Vorgaben bei der Verarbeitung personenbezogener Informationen z.B. nach dem BDSG [BDSG15], TMG [TMGe15] oder TKG [TMGe16] können verschiedene Maßnahmen umgesetzt werden. Im entwickelten System wurde als wesentliches Element die Pseudonymisierung von Benutzerkennungen und IP-Adressen genutzt. Kommerzielle SIEM-Systeme, die primär einem amerikanischen Verständnis von Datenschutz folgen, bieten hier oft keine Unterstützung, wenige zumindest die Möglichkeit der Anonymisierung oder Pseudonymisierung von Benutzerkennungen oder Email-Adressen.

Für IP-Adressen wurde deshalb ein eigenes Verfahren entwickelt, dass sowohl eine präfixerhaltende Umkodierung vornimmt (d.h. die IP-Adressen in einen anderen Adressbereich kodiert) als auch eine Pseudonymisierung realisiert, d.h. IP-Adressen durch ein symmetrisches Verschlüsselungsverfahren verschleiert. Einzelne markante bzw. allgemein bekannte IP-Adressen oder Adressbereiche ohne Personenbezug (z.B. Server-, Netzwerk- oder Broadcast-Adressen) können dabei ausgespart werden, um Analysen zu erleichtern. Abgesehen von der Umkehrbarkeit der Verschleierung und der Aussparung markanter Adressen ist das Verfahren mit den oft bei der Anonymisierung von Netzwerkprotokollen verwendeten Verfahren vergleichbar [FXAM04]. Für Zeichenketten wurde ein klassisches asymmetrisches Verschlüsselungsverfahren ohne Salt genutzt, d.h. eine Eingabe wird immer durch den gleichen verschlüsselten Text ersetzt. Durch den Einsatz dieser Verfahren können forensische Analysen und auch die Korrelation der Events für die SIEM-Regeln in pseudonymisierter Form erfolgen. Im Falle eines begründeten Verdachts können dann im Rahmen definierter Freigabeprozesse und Mitbestimmungsregeln die Originalwerte nach einem asymmetrischen Verfahren wiederhergestellt werden.

Es ist anzumerken, dass die implementierten kryptografischen Verfahren geschwächt werden mussten, um die Fähigkeit zur Korrelation zu erhalten. Für sich allein gesehen bilden sie keinen sicheren Schutz der Vertraulichkeit, da sie für Brut-Force-Angriffe anfällig sind. Sie werden aber mit klassischen Zugriffskontrollmechanismen kombiniert, die den Zugriff auf die verschleierten Daten auf wenige berechtigte Personen begrenzen. Zudem werden die verwendeten Schlüssel in regelmäßigen Abständen ausgetauscht. Für den berechtigten Personenkreis bildet die Verschleierung aber einen wirksamen Schutz, da personenbezogene Log-Informationen nicht einfach und schnell aus reiner Neugierde eingesehen werden können sondern es eines beträchtlichen Aufwandes bedarf, die Verschleierung zu brechen. Ein solches Verhalten kann dann ohne Zweifel als vorsätzliche Pflichtverletzung angesehen werden, die mit entsprechenden rechtlichen Konsequenzen geahndet werden kann.

4.3 Batch-Import und historische Korrelation

LM-Systeme können typischerweise sehr gut damit umgehen, dass Log-Informationen im Batchbetrieb und auch mit großem zeitlichem Versatz importiert werden. SIEM-Systeme arbeiten dagegen typischerweise im Echtzeitmodus, d.h. Events werden mit dem Zeitstempel des Empfangs im SIEM-System im Regelwerk verarbeitet und können nicht in der „Vergangenheit empfangen" werden. Im implementierten System werden die meisten Log-Informationen (nach dem Volumen gemessen) über Dateien im Batchbetrieb mit einem Versatz von mind. einer Stunde importiert, da die Quellen die Log-Dateien nur so bereitstellen. Deshalb muss das System sowohl bei der Verarbeitung mit einem hohen Burst-Aufkommen an Events arbeiten können, als auch Events in SIEM-Regeln korrelieren können, deren Zeitstempel in der Vergangenheit liegt („historische Korrelation"). Diese Fähigkeit ist bei kommerziellen Systemen nur begrenzt möglich.

Eine besondere Form des Batch-Imports sowie der historischen Korrelation stellen forensische Untersuchungen von Server- oder Client-Systemen dar. Im Zuge der Untersuchung werden sehr detaillierte Protokolle der Systemaktivitäten und –nutzung erstellt, die im Regelbetrieb nicht erfasst werden. Um diese Protokolle in den Kontext des Netzwerkes setzen zu können, müssen sie nachträglich mit einem zeitlichen Versatz von mehreren Tagen oder Wochen integriert und die Events des betroffenen Zeitraums nachträglich erneut korreliert werden können.

4.4 Architektur

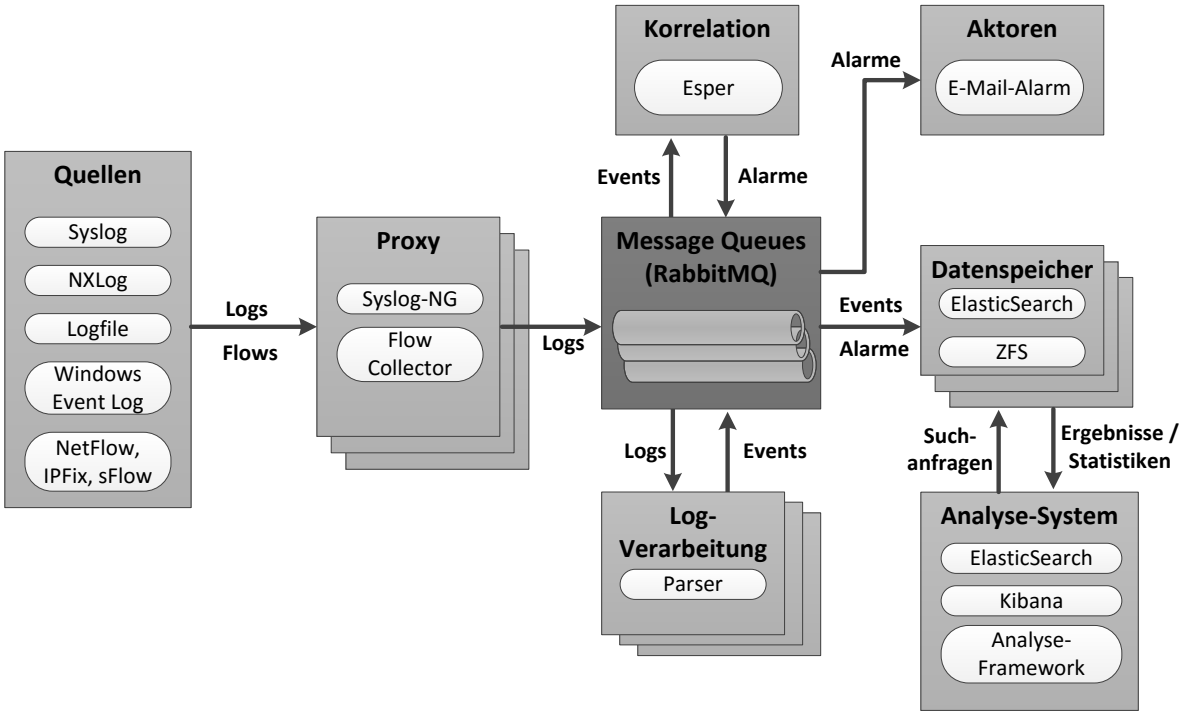

Abb. 1: Architektur des SIEM-Systems

In der Abbildung ist die Grundstruktur des entwickelten SIEM-System dargestellt. Für eine möglichst gute Skalierung ist das System verteilt ausgelegt. Einzelne Komponenten können flexibel zur Erhöhung der Verarbeitungsleistung oder Verfügbarkeit redundant ausgelegt werden. Die Kommunikation erfolgt innerhalb des Systems verschlüsselt und authentifiziert.

Als Quellen werden verschiedene Datenquellen unterstützt. Als Agenten werden bevorzugt die auf vielen Systemen bereits vorhandenen syslog-Komponenten genutzt. Auf Windows-Systemen werden Windows Event Logs entweder direkt über nxlog [Nxlog] im JSON-Format erhoben und per TLS weitergeleitet oder zuerst über Windows Event Forwarding verschlüsselt auf einem zentralen Windows-System gesammelt und von dort per nxlog weitergeleitet. Für Log-Dateien werden dabei verschiedene Übertragungsmodi unterstützt. Im Push-Modus werden die Log-Dateien durch die Quelle auf ein Proxy-System hochgeladen – entweder zeilenweise durch einen Agenten auf der Quelle, der die Log-Datei auf Veränderungen hin überwacht, oder im Batch-Betrieb durch einen periodischen Prozess, der die Log-Datei in festen Abständen komplett überträgt. Im Pull-Modus wird die Log-Datei ebenfalls im Batch-Betrieb in periodischen Abständen durch den Proxy von der Quelle abgerufen. Der Pull-Modus kommt auch bei anderen Quellen wie z.B. SQL-Datenbanken zum Einsatz, die periodisch durch einen Proxy abgefragt werden. Bei Benutzung des Batch-Betriebs werden die Log-Dateien auf dem Proxy jeweils in Log-Einträge zerlegt, die dann einzeln an die Log-Verarbeitung übergeben werden. Flow-Informationen über Netzwerkkommunikation (z.B. von Routern, Firewalls und Switches) werden im Binärformat an den Proxy übertragen, dort dekodiert und im Textformat wie reguläre Log-Einträge an die Log-Verarbeitung weitergeleitet.

Der Proxy ist eine leichtgewichtige Linux-Maschine, die verschiedene Aufgaben erfüllt. Da ein SIEM-System optimal in einer sehr gut gesicherten Netzwerkzone steht, müssen viele Log-Quellen aus weniger sicheren Zonen in diese senden, was nicht wünschenswert ist.

Deshalb sammelt der Proxy Log-Informationen innerhalb einer Zone und gibt diese über einen einzigen, verschlüsselten Datenkanal nach innen weiter. Er sorgt dabei auch dafür, dass syslog-Quellen, die keine TCP- oder TLS-Verbindungen unterstützen nur innerhalb einer Sicherheitszone unverschlüsselt kommunizieren, um so die Vertraulichkeit und Zuverlässigkeit zu verbessern. Der Proxy stellt zudem Pufferspeicher bereit, um im Falle einer Netzstörung oder Überlastung der Log-Verarbeitung Log-Informationen zwischenspeichern zu können.

Die SIEM-Komponenten kommunizieren untereinander über Message Queues. Hierfür wurde RabbitMQ ausgewählt, da es langfristig erprobt ist, sehr gute Performance-Eigenschaften aufweist und zudem keine JAVA-Laufzeitumgebung benötigt, die aus Sicherheitsgründen in der Betriebsumgebung auf möglichst wenigen Systemen genutzt werden sollte. Daneben bietet es verschiedene Möglichkeiten, den Betriebszustand in Echtzeit zu überwachen und dynamisch zu steuern. Der Einsatz von Message Queues bietet eine Reihe von Vorteilen gegenüber klassischer syslog- oder Client-Server-Kommunikation. Wesentlich für das entwickelte System ist die Möglichkeit, Metadaten an einzelne Messages (Log-Informationen) binden zu können, ohne diese in den Log-Eintrag kodieren zu müssen. So werden z.B. Informationen zur Identität der Quelle (wenn diese in den Log-Einträgen nicht enthalten ist) sowie zur Kodierung der Log-Daten transportiert. So können parallel unstrukturierte Texte, oder als XML, JSON, CVS oder KVP (Key Value Pairs) kodierte Log-Daten transportiert und bei der Log-Verarbeitung direkt der passende Parser ausgewählt werden. Hierdurch können auch einzelne Schritte der Log-Verarbeitung auf die Quelle bzw. den Proxy verschobene werden. So erzeugen einige Quellen wie Windows-Systeme mit nxlog bereits strukturierte Log-Einträge im JSON-Format. Diese Zerlegung muss bei der Log-Verarbeitung dann nicht noch einmal vorgenommen werden. Auch die Alarmmeldungen der Korrelation sind in JSON kodiert und können dann wie andere Log-Informationen weiter verarbeitet werden. Weitere wichtige Eigenschaften der Message Queues sind die Flusssteuerung, persistente Pufferung, Priorisierung und Filterung. Durch die Flussteuerung kann die Übertragungsrate zwischen einzelnen Knoten gesteuert werden, um deren Last und somit auch Verarbeitungslatenz besser kontrollieren zu können. In Überlastsituationen kann durch die Priorisierung zudem die Latenz zeitkritischer Meldungen verringert werden. Ebenfalls in Überlastsituationen, d.h. wenn Messages nicht direkt weitergeleitet werden können, werden sie durch die persistente Pufferung vor Verlust geschützt. Die Filterung wiederum sorgt dafür, dass jedem System nur die für dieses relevanten Messages zugestellt werden, dies aber flexibel gesteuert werden kann, um z.B. einzelne Verarbeitungskomponenten flexibel zwischen den Maschinen des SIEM-Systems verschieben zu können. Dazu kann die Kommunikation komplett verschlüsselt erfolgen und Knoten müssen sich vor dem Senden oder Empfangen authentifizieren. RabbitMQ kann selber als verteiltes System oder Cluster arbeiten und hierdurch sehr weit skalieren bzw. auch hochverfügbar ausgelegt werden [Kuch14].

Die Log-Verarbeitung selber besteht aus einer Vielzahl von Parser-Modulen, die je nach Typ die Log-Einträge zerlegen, Felder extrahieren und normalisieren, die Meldungen kategorisieren und ggf. die Zeichenkodierung vereinheitlichen. Da die Log-Verarbeitung auf den Message Queues arbeitet, kann das System durch Hinzufügen weiterer, voneinander unabhängiger, Instanzen linear skaliert werden. Hierdurch vereinfacht sich auch die Implementierung der Log-Verarbeitung, da keine Aspekte wie Nebenläufigkeit oder Synchronisation berücksichtigt werden müssen bzw. durch RabbitMQ realisiert werden. Des Weiteren können durch diese Entkopplung Parser-Module in unterschiedlichen Programmiersprachen entwickelt werden. In der aktuellen Implementierung wurden alle Module in Perl entwickelt, da es eine sehe gute Performance bei der Auswertung regulärer Ausdrücke (zum Zerlegen der Log-Informationen) bietet und als Skriptsprache eine schnelle Entwicklung ermöglicht. Für die Optimierung des Durchsatzes können Parser für einzelne Quellen mit einem sehr hohen Aufkommen an Log-Informationen (z.B. Firewall oder Web Proxy)

aber auch z.B. in C++ entwickelt werden. Grundvoraussetzung ist lediglich, dass eine passende Client-Bibliothek für RabbitMQ bereitsteht, was für viele Programmiersprachen der Fall ist.

Am Ende der Log-Verarbeitung werden alle Events als JSON kodiert und in einem ElasticSearch-Cluster gespeichert. ElasticSearch [Elasti] wurde als Datenspeicher gewählt, da es sehr gut skaliert und umfangreiche Such-, Filter- und Aggregationsfunktionen bereitstellt. ElasticSearch lässt sich nicht nur zu einem sehr großen Cluster ausbauen, es erlaubt diesen Umbau auch zur Laufzeit, d.h. im laufenden Betrieb können weitere Knoten dem Cluster hinzugefügt werden und die Last wird dynamisch auf diese verteilt. Wird ein Knoten entfernt, wird dessen Last auf andere Knoten verteilt. Dies gilt insbesondere für die redundante Speicherung von Daten, d.h. fällt ein Knoten aus, so ist für einen Teil der Suchindizes die Redundanzanforderung nicht länger erfüllt und der Cluster beginnt automatisch damit, die verbliebenen Instanzen im Cluster zu replizieren, bis die geforderte Redundanz wieder gewährleistet ist. Als Benutzerschnittstelle zur interaktiven Suche in den Log-Daten, zur Visualisierung von Suchergebnissen sowie zur Realisierung von Dashboards wird Kibana genutzt, dass parallel mit ElasticSearch entwickelt wird. ElasticSearch bietet zwar die Möglichkeit, Daten im Suchindex zu komprimieren, die erreichten Kompressionsraten sind aber gering. Dies liegt primär daran, dass ElasticSearch jeden Log-Eintrag als ein Dokument betrachtet und einzeln komprimiert. Deshalb werden die Suchindizes auf einem ZFS-Dateisystem mit aktivierter Kompression gespeichert, was zu deutlich höheren Kompressionsraten führt.

Für die Arbeit mit den Log-Daten existiert ein Analyse-System. Dies ist ein Linux-System, auf dem eine lokale ElasticSearch-Instanz mit Kibana sowie die Werkzeuge zur Nachverarbeitung der Suchergebnisse laufen. Bei komplexen Analyseaufgaben wird ein Skript auf dem Analyse-System gestartet, dass zunächst eine Suche auf dem Datenspeicher ausführt und die Ergebnisse danach in einer Pipeline mit verschiedenen Operatoren nachverarbeitet. Die Nachverarbeitung umfasst dabei Filter, Feldtransformationen, Kalkulationen oder weitere Aggregationen. Auch die Anreicherung der Ergebnisse mit weiteren Informationen (z.B. Reputation oder Geo-Lokationen zu IP-Adressen) sowie der JOIN zwischen mehreren Ergebnissen ist hier möglich. Die Ausgabe erfolgt dann in eine Datei oder das lokale ElasticSearch, von dem aus die Ergebnisse mittels Kibana visualisiert werden können. Derartige Analyse-Skripte können bei Bedarf oder zeitgesteuert gestartet werden. Bei Bedarf kann die Pipeline durch eigene Operatoren erweitert werden. Auch eine mehrstufige Weiterverarbeitung mit dem lokalen ElasticSearch als Zwischenspeicher ist möglich, um beliebige Forensik-Aufgaben abbilden zu können. Aktuell werden alle Operatoren in Perl implementiert. Prinzipiell kann die Nachverarbeitung in jeder Programmiersprache implementiert werden, die entweder über einen Client für ElasticSearch verfügt oder direkt mit dem bereitgestellten REST-Interface umgehen kann.

Zur Realisierung der SIEM-Funktionalität, d.h. für die Korrelation der Events wird die Open-Source-Version von Esper [Esper], einer JAVA-Bibliothek für Complex Event Processing (CEP), benutzt. Esper erlaubt es, einen Strom von Events mit Regeln in EPL (Event Processing Language), einer SQL-Variante für Datenströme, hinsichtlich auffälliger Muster zu analysiert. Der wesentliche Unterschied zu SQL ist die Definition von Fenstern bestimmter Größe (in Events) bzw. über die Zeit, in denen die SQL-Statements angewendet werden. Esper bzw. EPL bieten auch eingebaute Sprachelemente, um die Ausführung der Auswertung automatisch zu parallelisieren. In der entwickelten Architektur läuft die SIEM-Engine dabei als eigenständige Multithreading-Anwendung. Auf der Eingangsseite arbeiten mehrere Threads als RabbitMQ-Clients, d.h. sie lesen kontinuierlich die in der Log-Verarbeitung erzeugten, normalisierten Events im JSON-Format ein. Hierbei wird die Fähigkeit von RabbitMQ genutzt, bei Bedarf Events automatisch zu duplizieren, um sie an mehrere Empfänger zu senden. Die Module für die Log-Verarbeitung sowie Speicherung in ElasticSearch müssen dafür nicht modifiziert werden. Die so eingelesenen Events werden in der Esper-Bibliothek mit den definierten Regeln ausgewertet. Werden entsprechende Muster erkannt,

so wird ein synthetischer Alarm-Event erzeugt und dieses an RabbitMQ zurück übergeben. Der Alarm-Event referenziert dabei die für die Auslösung des Alarms relevanten Events. Ein Event kann im Zuge der Auswertung auch durch mehrere Regeln parallel als auffällig erkannt und gemeldet werden. Regeln in EPL können dabei sowohl trivial sein wie das Erkennen eines einzelnen Events eines Schadcode-Fundes aber auch komplex wie das Erkennen einen Portscans über unterschiedliche Systeme des Netzwerkes hinweg oder gar ganze Sequenzen eines typischen Angriffsmusters (z.B. von einer Phishing-Mail bis zur Kompromittierung eines Systems). Die an RabbitMQ zurückgelieferten Alarm-Events werden in den Metadaten besonders gekennzeichnet. Die Alarm-Events werden dann ebenfalls in ElasticSearch gespeichert.

Eine besondere Herausforderung bei der Entwicklung eines Regelwerkes für Esper ist die Testbarkeit der Regeln. Für einfache Regeln, d.h. Regeln die auf Einzelevents prüfen, oder Regeln, die nur auf einem sehr kurzen Zeitfenster operieren, bietet es sich an, einen Generator zu implementieren, der die Events zu vordefinierten Zeitpunkten an Esper übergibt. Sollen dagegen Regeln getestet werden, die über sehr lange Zeitfenster operieren, bietet es sich an, die Möglichkeit zur historischen Korrelation zu nutzen. Dabei wird der Zeitstempel eines Events nicht aus der aktuellen Zeit bestimmt sondern aus einem definierten Feld des Events. Wichtig hierbei ist, dass die Events zeitlich sortiert an Esper geliefert werden, da die interne Zeit nur monoton steigen kann.

Parallel können aber auch andere Programme (Aktoren) aktiv auf das Eintreten eines Alarms warten und darauf reagieren, z.B. durch Starten eines Programms oder Versenden einer E-Mail. Hierdurch können auch Aktionen zur Automatisierung der Behandlung von Sicherheitsvorfällen bzw. durch Umsetzung von temporären Gegenmaßnahmen (z.B. Sperrung von Accounts bzw. Entzug kritischer Berechtigungen bei Verdacht auf Account-Missbrauch) implementiert werden. Eine ausführliche Behandlung dieser Thematik findet sich in [MaSu15]. Bei der Implementierung der Aktoren wird wiederum die Fähigkeit von RabbitMQ genutzt, Events bei Bedarf automatisch zu duplizieren.

4.5 Erfahrungen

Zur besseren Bewertung, ob eine Eigenentwicklung eine kostengünstige Option sein kann, sollen zunächst die Erfahrungen zu wesentlichen Kostentreiber betrachtet werden. Eine wesentliche Anforderung war eine sehr gute Skalierbarkeit mit einem Durchsatz von über 10.000 Events pro Sekunde (EPS). Um diese Werte zu erreichen, waren umfangreiche Tests notwendig, um einerseits die Flaschenhälse in der Software zu erkennen und andererseits den Einfluss der Hardwareleistung im Detail zu verstehen. Die hatte verschiedene Umbauten in der Architektur die Folge. Für ein kleines System mit bis zu 1000 EPS wäre dieser Aufwand weitestgehend entfallen. Weitere Tests waren notwendig, um die Kompatibilität verschiedener Softwarekomponenten und Programme zu bestimmen. So funktionieren zwar die meisten Agenten und syslog-Server reibungslos zusammen solange unverschlüsselt über UDP kommuniziert wird, soll aber zum Schutz von Integrität und Vertraulichkeit TLS-Verschlüsselung genutzt werden, zeigen sich gravierende Kompatibilitätsprobleme. Ähnlich verhält es sich mit RabbitMQ und verschiedenen Client-Implementierungen. Nach der Festlegung und Test der Basisarchitektur und Datenübertragung bestand der zweite große Aufgabeblock aus der Implementierung der Normalisierung. Hier ist es von Vorteil, wenn die zu überwachende IT-Landschaft möglichst homogen ist, da die Lizenzkosten kommerzieller Systeme mit dem Durchsatz ansteigen, die Kosten der Eigenentwicklung primär mit der Vielfalt der Systeme. Der dritte große Aufgabenblock bestand aus der Entwicklung von Reports bzw. Dashboards, Analysefunktionen und SIEM-Regeln. Dieser Teil kann beliebig aufwändig werden. Ein kommerzielles System hat hier aber nur Vorteile, wenn die gewünschten Funktionen bereits im Produkt vorliegen. Typischerweise müssen auch hier umfangreiche Anpassungen vorgenommen werden. Die Kosten für die Konzeptionierung des Systems, Hardware, Betrieb sowie die Anpassung und Pflege der Datenquellen sind vergleichbar. Der Aufwand für Pflege und Weiterentwicklung der Software sind bei

der Eigenentwicklung typischerweise höher als bei kommerziellen Lösungen, dafür entfallen die entsprechenden Lizenzkosten. Insgesamt ergibt sich der größte Kostenvorteil bei einem System, dass primär für das Log Management bzw. die Forensik genutzt wird und das ausgewählte sicherheitskritische Vorgänge über Regeln automatisch überwacht.

Die in Abbildung 1 dargestellte Architektur stellt das Ergebnis mehrerer Entwicklungsstufen dar. In früheren Versionen des Systems wurden für die Kommunikation zwischen den Komponenten klassische UNIX-Interprozesskommunikation, Shared Memory und Remote Procedure Calls (RPC) genutzt. Dabei zeigte sich, dass insbesondere die Skalierung der Log-Verarbeitung sehr komplex ausfällt und sehr viel Zeit für die Synchronisation verloren ging. RabbitMQ zeigte hier deutliche Vorteile und vereinfachte die Implementierung der Komponenten signifikant. Der Einsatz von Message Queues vereinfachte die Implementierung auch dahingehend, dass die verschiedenen Funktionalitäten stark voneinander entkoppelt werden konnten. So können weitere Aktoren, die aktiv auf das Erzeugen eines Alarms reagieren als Clients an RabbitMQ gebunden werden und von dort die Alarm-Events empfangen und eigenständig verarbeiten und darauf reagieren. Ebenso könnten auch weitere Analysemodule implementiert werden, die Events mit anderen Methoden als den EPL-Regeln in Esper analysieren.

RabbitMQ bietet nicht nur Vorteile, es zeigt auch Eigenarten, auf die an dieser Stelle hingewiesen werden soll. RabbitMQ beinhaltet eine Reihe von Mechanismen, um in Situationen mit knappen Systemressourcen durch eine Flusskontrolle eine Überlast zu vermeiden. Diese Mechanismen sind aber primär darauf ausgelegt, dass viele Sender und Empfänger auf vielen getrennten Message Queues arbeiten und die Queues nicht als Pufferspeicher gebraucht werden. Werden nur wenige oder gar nur eine Message Queue verwendet, so führen die Flow-Control-Mechanismen und der interne Garbage Collector zu einem sehr ungleichmäßigen Datenfluss und einem suboptimalen Durchsatz. Durch die Verwendung von mehreren gleichartigen Queues im Round-Robin-Verfahren kann der Durchsatz signifikant gesteigert und gleichmäßiger gestaltet werden. Auf einem 4-Kern-System mit 8 GB Speicher kann so ein Durchsatz von >10.000 Events pro Sekunde erreicht werden (ca. 1 KB je Event). Besteht die Möglichkeit, die Senderate der Quellen (z.B. beim Einlesen von Log-Dateien im Batchbetrieb) zu kontrollieren, so sollte diese Option genutzt werden, da sie effektiver ist, als die automatischen Flow-Control-Mechanismen. Insgesamt lässt sich der Durchsatz von RabbitMQ durch mehr Ressourcen (CPU, RAM, Disk-IOPS) deutlich steigern.

ElasticSearch hat sich als eine sehr effektive Suchmaschine erwiesen. In einem typischen ELK-Stack (ElasticSearch als Datenbank, LogStash zur Log-Verarbeitung und Kibana als User-Interface) wird ein Suchindex je Tag erzeugt. Werden viele verschiedene Log-Quellen mit vielen unterschiedlichen Datenfeldern (z.B. Windows Event Log) angebunden, erzeugt das einen speichertechnisch sehr ineffizienten Suchindex, da auch nicht vorhandene Felder eines Events Speicherplatz in einem Index verbrauchen. Deshalb ist es empfehlenswert, die Suchindizes primär nach Quelltypen und bei Bedarf ggf. noch nach Tagen zu gliedern. Insbesondere wenn eine Aufteilung nach Tagen genutzt wird, sind sehr viele Suchindizes parallel aktiv. ElasticSearch teilt diese weiter in sogenannte Shards auf, um den Zugriff auf einen Index parallelisieren zu können. Weitere Shards entstehen, wenn für eine verbesserte Verfügbarkeit Indizes redundant gespeichert werden. Per Default nutzt ElasticSearch 5 Shards und 2 Repliken, d.h. insgesamt 10 Shards pro Index. Da für jeden Shard ein Thread im Betriebssystem genutzt wird, ist dies solange von Vorteil, wie hinreichend viele CPU-Kerne im Cluster zur Verfügung stehen, um die Threads, die den bei einer Suchoperation angesprochenen Shards zugeordnet sind, auch parallel ausführen zu können. Es ist dabei zu beachten, dass auch die Threads für die Shards, in die laufend neue Events gespeichert werden, parallel aktiv sein müssen. Werden mehr Shards parallel angesprochen, als CPU-Kerne verfügbar sind, so sinkt der ElasticSearch-Durchsatz sogar wieder. Aus diesem Grund macht es in den meisten Fällen bei der oben beschriebenen Aufteilung keinen Sinn, pro Index mehr als einen Shard zu verwenden.

Eine weitere Herausforderung entsteht durch die Typisierung der Event-Felder in ElasticSearch. Für die Analyse und das Reporting ist es sehr hilfreich, wenn Zahlen oder IP-Adressen als ein entsprechender Datentyp und nicht als String gespeichert werden. Dadurch ist es dann z.B. möglich, IP-Adressen mit Netzmasken zu vergleich oder Zahlenwert zu aggregieren. Und auch bei Strings stellt sich die Frage, ob der gesamte String als ein Token betrachtet oder vorher in Worte zerlegt werden soll. Bei Freitextmeldungen ist meist eine Zerlegung wünschenswert, bei kurzen Fehlermeldungen wie „Login Denied" eher nicht, um später Aggregationen mit einer Zählung der Anzahl solcher Events durchführen zu können. Um das Verhalten von ElasticSearch besser steuern zu können, müssen die konkreten Datentypen vorab bei der Konfiguration eines Index mitgeteilt werden. Ein weiteres Problem entsteht, wenn Felder unterschiedlicher Quellentypen den gleichen Namen tragen aber unterschiedliche Datentypen aufweisen. Bei derartigen Konflikten verweigert ElasticSearch die Speicherung der Events. Zur Lösung dieser Problematik wurde wie folgt verfahren. Per Default werden Felder als String gespeichert und in Tokens zerlegt, um eine Volltextsuche zu ermöglichen. Felder werden durch die Nutzung von hierarchischen Feldnamen in einen eigenen Namensraum je Quellentyp gelegt, um Typkollisionen zu vermeiden. Für einen festgelegten Satz normalisierter Felder (z.B. Ereigniszeit, Quell- und Zieladresse) wurden die Typen vorab festgelegt und die Feldnamen und Datentypen werden in der Log-Verarbeitung entsprechend angepasst. Die Felder eines Quellentyps, die für das Reporting benötigt werden, werden bei der Anbindung der Quelle bestimmt und entsprechende Typendefinitionen in ElasticSearch vorgenommen.

Eine Kompression im Filesystem (ZFS mit maximaler GZIP-Kompression) hat sich als deutlich effizienter erwiesen, als die Kompression in ElasticSearch selber. ElasticSearch hat in den Tests mit Echtdaten Kompressionsraten von 1,2..1,4:1 erzielt, die ZFS-Kompression dagegen 5..8:1, da jeweils nicht nur einzelne Datenfelder sondern große Blöcke der Datenbank komprimiert werden. Der Performanceverlust (Durchsatz indexierter Events pro Sekunde) durch die ZFS-Kompression betrug in den Tests 10-15% auf einer 8-Kern-Maschine, wobei sich die CPU-Auslastung durch die Kompression um ca. 50% erhöhte. Die höhere CPU-Last ist aber auch bei Suchvorgängen zu beachten, da die Daten beim Lesen jeweils wieder dekomprimiert werden müssen. Stehen ausreichend CPU-Ressourcen bereit, sorgt die Kompression bei langsamen Speichermedien sogar für einen signifikanten Performancegewinn.

5 Ausblick

Die aktuelle Implementierung unterstützt noch nicht die von Esper angebotene historische Korrelation. Hierfür ist vorgesehen, nach einem Batch-Import mit Events in der Vergangenheit, alle Events des relevanten Zeitraums (Zeitraum der Batch-Events zzgl. der entsprechenden Zeitfenster des Regelwerkes) aus ElasticSearch zu extrahieren und durch eine eigene Esper-Instanz zu leiten und mit den Alarmen der Echtzeitinstanz zu kombinieren. Dies soll insbesondere den Import von forensischen Zeitreihen-Analysen (z.B. aus Plaso / Log2Timeline [Plaso]) ermöglichen, um so die Protokollierung einzelner Systeme auch nachträglich noch zu ermöglichen. Praktisch sollen so Erkenntnisse aus der forensischen Analyse eines kompromittierten Systems in den Kontext des gesamten Netzwerkes eingebettet werden.

Für die Weiterentwicklung ist ebenfalls geplant, die einzelnen SIEM-Komponenten in Docker-Container zu verpacken, um sie flexibler auf unterschiedlichen Systemlandschaften mit unterschiedlichen Skalierungsanforderungen ausrollen und zur Laufzeit umkonfigurieren zu können.

Literatur

[BSI03] BSI: Studie über die Nutzung von Log- und Monitoringdaten im Rahmen der IT-Frühwarnung und für einen sicheren IT-Betrieb. (2013)

[Splu16] Splunk Inc.: Splunk Search Reference. http://docs.splunk.com/Documentation/Splunk/latest/SearchReference (2016)

[DuKr14] Düsseldorfer Kreis: Orientierungshilfe zu den Datenschutzanforderungen an App-Entwickler und App-Anbieter. (2014)

[DuKr09] Düsseldorfer Kreis: Orientierungshilfe Protokollierung. (2009)

[BDSG15] Bundesdatenschutzgesetz (BDSG). https://www.gesetze-im-internet.de/bdsg_1990/, In: BGBl. I S. 162 (2015)

[TMGe15] Telemediengesetz (TMG). https://www.gesetze-im-internet.de/tmg/, In: BGBl. I S. 1324 (2015)

[TMGe16] Telekommunikationsgesetz (TKG). https://www.gesetze-im-internet.de/tkg_2004/, In: BGBl. I S. 1217 (2016)

[FXAM04] J. Fan, J. Xu, M. H. Ammar, S. B. Moon: Prefix-Preserving IP Address Anonymization. In Computer Networks, Elsevier Verlag, Volume 46, Issue 2 (2004)

[Nxlog] NXLog Community Edition, http://nxlog.org/

[Rabbit] RabbitMQ Homepage, http://www.rabbitmq.com/

[Kuch14] Jerry Kuch: RabbitMQ Hits One Million Messages Per Second on Google Compute Engine. https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine (2014)

[Elasti] ElasticSearch Homepage, http://elasti.co/

[Esper] EsperTech Homepage, http://espertech.com/

[Plaso] Plaso Homepage, https://github.com/log2timeline/plaso/wiki

[MaSu15] D. Mahrenholz, R.Schumann: Incident Response im SIEM-Kontext – Ein Erfahrungsbericht, In: P. Schartner, P. Lipp. „DACH Security 2015", syssec (2015)