Seminararbeit

Service und Security-Monitoring

Seminar: Aufbereitung und Auswertung komplexer Daten

Martin Steinbach
Juni 2018

universität Rostock

Exzerpt

Servicemonitoring ist eine wichtige Voraussetzung um eine zuverlässige IT-Infrastruktur zu betreiben. Monitoring ist auch geeignet um IT-Sicherheitskritische Ereignisse zu identifizieren und adäquat auf diese zu reagieren. Die vorliegende Arbeit bietet eine Einführung in die Thematik der Dienstüberwachung und stellt die beiden Überwachungsformen Aktive- und Passive-Überwachung vor. Es wird zudem die Frage geklärt, warum Servicemonitoring auch gleichzeitig Securitymonitoring ist. Anschließend wird anhand eines existierenden Prototypen aufgezeigt, wie eine Korrelation von Ereignissen in Cloud-Umgebungen realisiert werden kann. In Abschnitt 3 wird eine mögliche Visualisierungslösung für die korrelierten Daten vorgestellt. Die letzte Sektion zieht ein Fazit und erläutert zukünftige Ideen für die Entwicklung des Prototypen.

Inhaltsverzeichnis

1 Einführung

In diesem Kapitel wird anhand der IT-Sicherheitsziele aufgezeigt, dass man unter dem Begriff Servicemonitoring auch immer den Begriff Securitymonitoring verstehen kann. Auch soll darauf hingewiesen werden, dass der Ausdruck Überwachung im ganzen Dokument mit der Bedeutung: Aufsicht oder Monitoring belegt wird um eine klare Abgrenzung zur zweiten Bedeutung: Observation, Beschattung (engl. surveillance) zu erlangen.

1.1 Servicemonitoring und Securitymonitoring

Die Überwachung von Diensten ist mittlerweile ein integraler Bestandteil der Infrastruktur jedes IT-Diensteanbieters geworden. Neben der einfachen Erfassung und der (z.B. grafischen) Aufarbeitung verschiedenster Messgrößen, werden die erfassten Daten zunehmend analysiert und es wird versucht Muster zu erkennen. Dieser Vorgang wird auch als *BigData* bezeichnet. Diese Daten werden auch verstärkt zur Sicherheitsanalyse herangezogen. Daher stellt sich die Frage, ob Securitymonitoring äquivalent zum Servicemonitoring - Begriff ist. Um es vorweg zunehmen, ja, denn es kommt ausschließlich auf die Fragestellung an, die man mit den erfassten Daten klären möchte. Im Folgenden werden die drei Hauptziele der IT-Sicherheit aufgeschlüsselt und in Beziehung mit dem Servicemonitoring - Begriff gebracht.

Vertraulichkeit

Das Ziel der Vertraulichkeit sagt aus, dass der Zugriff auf Daten ausschließlich von autorisierten Nutzern erfolgen darf, egal in welchem Modus. Erreicht wird das Ziel zum Beispiel durch Zugriffsrechte, wie z.B. Mandatory Access Control (MAC)[1] oder Discretionary Access Control (DAC)[2] und vor allem durch Verschlüsselung.

Die Frage, ob sich Vertraulichkeit überwachen lässt, ist nur teilweise beantwortbar. Stellt man sich ein System vor auf dem ein nicht autorisierter Nutzer Zugriff auf Informationen erlangt, so ist dies messbar und es ist möglich eine Meldung zu generieren (z.B. eine Log-Meldung oder eine Nachricht an Verantwortliche). Wird jedoch ein autorisiertes Konto durch einen nicht autorisierten Nutzer kompromittiert, gestaltet sich die Entdeckung dieses Ereignisses schwieriger. Ob es sich in diesem Fall um einen erlaubten Zugriff des tatsächlichen Nutzers oder einen nicht erlaubten Zugriff handelt kann nur unter der Zuhilfenahme weiterer Information geklärt werden. Zum Beispiel könnte die Quelle, von der aus sich der Nutzer Zugriff verschafft hat, miteinbezogen werden. Auch die Korrelation mit Zeitdaten, an denen sich der zugriffsberechtigte Nutzer einloggt, können zur Klärung hinzugezogen werden.

Verfügbarkeit

Ob ein Dienst verfügbar ist, wird dadurch geklärt, ob der Zugriff auf Informationen innerhalb eines gewissen Zeitraums erfolgreich ist.

Die Verfügbarkeit gleicht damit auch der grundlegenden Fragestellung des Servicemonitorings. Ist ein gewisser Dienst erreichbar und ist dessen Abarbeitungsgeschwindigkeit in einem vorgegebenem Rahmen?

[1] Zugriffskontrollstrategie aus systemweiten Regeln, bei der nicht nur die Nutzeridentität ein Rolle spielt.
[2] Zugriffskontrollstrategie, bei der lediglich die Nutzeridentität eine Rolle spielt (übliches Rechtesystem).

Integrität

Integrität wird erreicht, wenn eine Änderung der Daten nicht unbemerkt geschieht. Es soll somit ein Indikator für die Veränderung existieren. Um dieses Ziel zu erreichen, werden Verfahren wie digitale Signatur und Hashes verwendet.

Auch das Ziel der Integrität lässt sich kontrollieren. Dazu finden die selben Maßnahmen Verwendung wie in der IT-Sicherheit. Es lassen sich zum Beispiel auf regelmäßiger Basis Daten prüfen, von denen man vorher mit einem kryptografisch sicheren Verfahren ein Hash errechnet hat. Ändert sich die Hashsumme, ohne dass ein Zugriff auf die Daten genehmigt wurde, ist dies ein Integritätsverlust.

Zusammengefasst ist Monitoring von IT-Infrastruktur immer auch gleich Securitymonitoring. Mithilfe eines lückenlos ausgerollten Monitorings ist es demnach möglich einen Beweis zu führen, zu welchem Zeitpunkt ein gewisser Dienst welchen Zustand hatte.

1.2 Motive

Warum eine dauerhafte Überwachung von Infrastruktur und den darauf aufbauenden Diensten keine Option sondern obligatorisch sein sollte, ist recht simpel zu erörtern. Allein die in [1, Seite 461] berichteten Zahlen sprechen für sich. 90 % aller Firmen waren schon Cyberattacken ausgesetzt, 80 % davon haben dadurch erhebliche finanzielle Einbußen erlitten. Aktuell werden innerhalb eines Jahres 86 % der großen nordamerikanischen Unternehmen Opfer von Cyberattacken und der Diebstahl des geistigen Eigentums hat sich in den Jahren 2011-2015 verdoppelt.
Auch der aktuelle jährlich veröffentlichte Lagebericht zur nationalen IT-Sicherheit des Bundesamtes für Sicherheit in der Informationstechnik (BSI) [2, Seite 12] berichtet von einer Cyberattacke auf einen großen deutschen Industriekonzern. Etwa **zwei Monate** konnten unbemerkt Daten aus weltweit verteilten Standorten in Richtung Südostasien abfließen, bevor der Vorfall detektiert wurde. Aus den Empfehlungen des BSI lässt sich schließen, dass neben einer ungünstigen Netzwerksegmentierung auch mangelhaftes Monitoring der Grund für die späte Erkennung war. In diesem Zusammenhang ist auch der Angriff auf den Deutschen Bundestag im Jahr 2015 erwähnenswert, da auch in diesem Fall einige Wochen lang unbemerkt Daten abfließen konnten und das Ziel hoheitlich war, Technologietransfer und finanzielle Absichten also eine untergeordnete Rolle gespielt haben.

1.2.1 Behörden mit Überwachungsauftrag

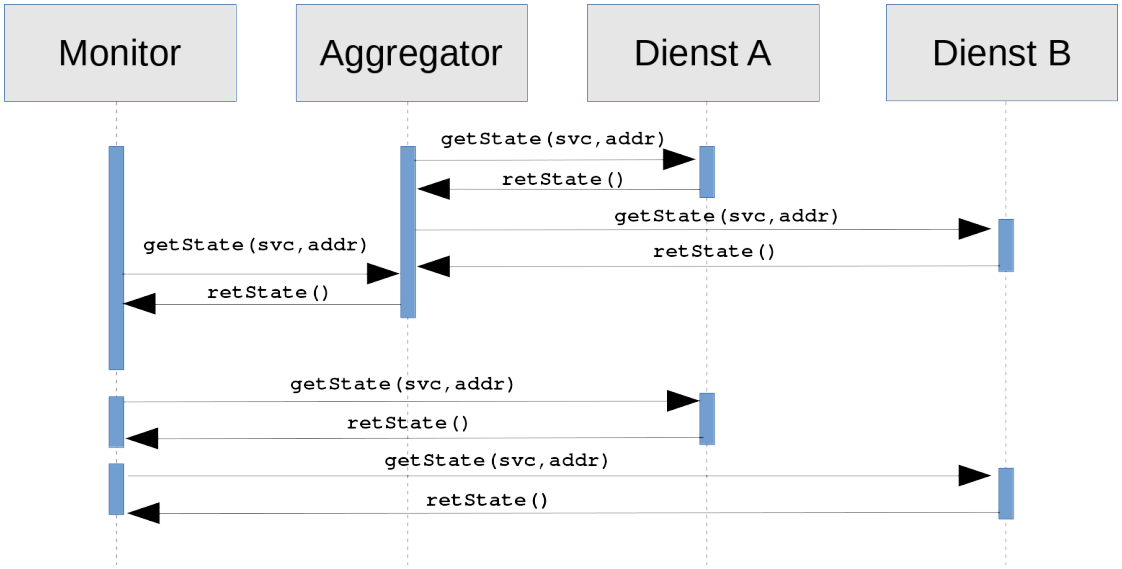
Aufgrund der zuvor dargestellten Gründe, wurden in den letzten Jahren eine Reihe an neuen Behörden in der Bundesrepublik Deutschland gegründet, deren Auftrag die Überwachung (Monitoring) wichtiger Infrastrukturen innerhalb der Grenzen der Bundesrepublik ist. Zum Einen das Nationale Cyber-Abwehrzentrum (NCAZ) [3] mit Sitz in Bonn. Die Aufgabe des NCAZ ist die Koordinierung von Abwehrmaßnahmen und Informationskonsolidierung über den Aufbau von rein ziviler Infrastruktur über mehrere Behörden hinweg. Das Nationale IT-Lagezentrum [4] überwacht hingegen aktiv die Regierungsnetze und erstellt monatliche Lageberichte. Auf militärischer Seite übernimmt der Bundeswehr-Organisationsbereich Cyber- und Informationsraum (CIR) diese Aufgabe.

1.3 Überwachungsformen

Es lassen sich zwei verschiedene Überwachungsformen identifizieren. Die aktive Überwachung, bei der aktiv Status und Messwerte von Diensten erfragt werden, sowie die passive Überwachung, bei der die Dienste selbstständig Meldungen an einen zentralen Punkt, den Monitor, senden. Unter einem Dienst wird in diesen Beispielen nicht nur ein *service* sondern jegliche Entität, deren Status messbar ist, verstanden.

1.3.1 Aktive Überwachung

Abbildung 1.1: Sequenzdiagramm: Aktive Überwachung



Obiges Sequenzdiagramm (Abbildung 1.1) zeigt schematisch den Ablauf von aktiver Überwachung. Der *Monitor* ist die zentrale Instanz auf der alle zu überwachenden Informationen gesammelt, aufbereitet und ausgegeben werden. Jedoch muss der *Monitor* nicht alle Informationen sammeln, es können auch hierarchisch untergeordnete *Aggregatoren* existieren, welche ebenfalls Informationen von verschiedenen Diensten erheben. Diese *Aggregatoren* können aus Leistungsgründen vor einen *Monitor* geschaltet werden, um die Anzahl an abzufragenden Diensten für den *Monitor* zu verringern. In diesem Fall bereitet bereits der *Aggregator* die Daten auf und der *Monitor* fragt nur noch die schon konsolidierten Informationen ab. Aber auch Segmentierungen von Infrastruktur können diesen Aufbau notwendig machen, wenn z.B. der *Monitor* *Dienst A* und *Dienst B* nicht direkt erreichen kann oder darf. Zur Klassifizierung von Ereignissen werden in Überwachungslösungen[3] oft verschiedene Status verwendet, dies dient hauptsächlich zum schnelleren Verständnis für die auswertende Person. Aus diesem Grund wurden in Abbildung 1.1 die Bezeichnungen für der Abfragefunktionen `getState()` und `retState()` gewählt. Mögliche Status für die Rückgabe sind in Tabelle 1.1 aufgeführt.

Tabelle 1.1: Statusübersicht

Statusbezeichnung	Statusbeschreibung
`OK`	Dienst läuft innerhalb normaler Parameter.
`WARNING`	Die (zuvor definierte) Warnschwelle wurde überschritten.
`CRITICAL`	Die kritische Schwelle wurde überschritten oder es gab einen Timeout.
`UNKNOWN`	Ein undefinierter Wert wurde an *Monitor* übermittelt.

Ein Visualisierungsbeispiel für einen Statusverlauf eines überwachten Dienstes befindet sich im Anhang (Abbildung 5.1).

[3] zum Beispiel: Nagios, Icinga

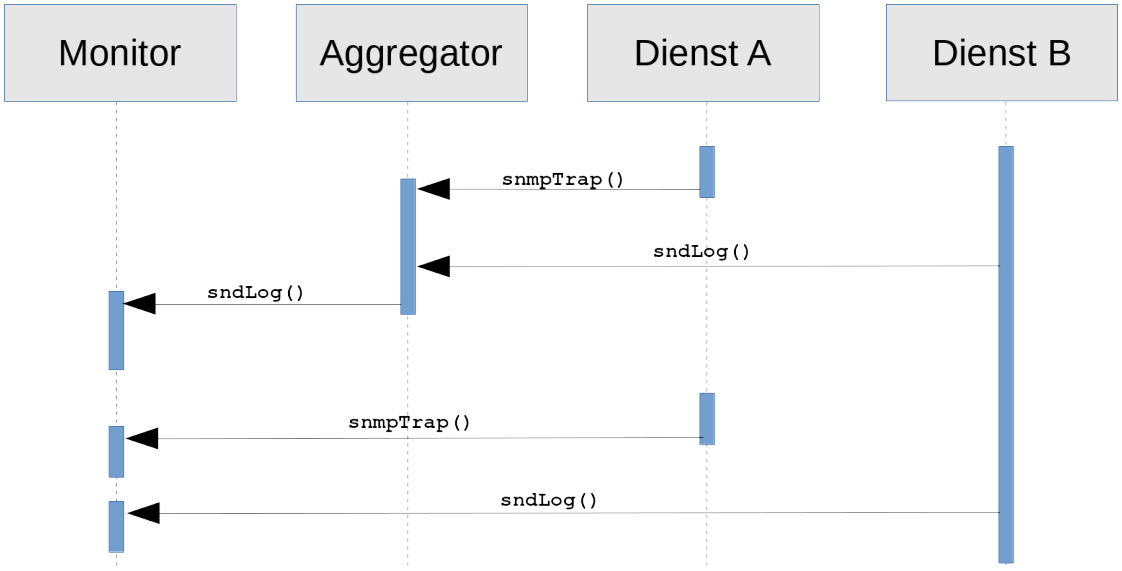
Mithilfe der Techniken zur aktiven Überwachung lassen sich zwei Klassen überwachbarer Dienste identifizieren: Die Betriebssystemabhängigen und die Betriebssystemunabhängigen Dienste. Tabelle 1.2 listet einige Beispiele für die jeweilige Klasse auf.

Tabelle 1.2: Beispiele für aktive Überwachung

Dienst / Entität	**Beispiel**
Betriebssystemabhängig	
Auslastung	wie viel CPU-Zeit benötigt ein bestimmter Prozess/das ganze System
Speicher	Speicherauslastung des Systems/ Belegung persistenter Speicher
Prozesse	läuft ein bestimmter Prozess/ wie viele Prozesse eines Namens laufen
Datendurchsatz	wie viele Bytes passieren ein Interface, Anzahl an *paket-drops*, *rejects*
Audit	wurden Zugriffsregeln verletzt/ welcher Nutzer hat auf Datei X zugegriffen
Betriebssystemunabhängig	
`ICMP`	Netzwerkschnittstelle/ System erreichbar, *round trip time*
`TCP/UDP`	ist bestimmter *port* erreichbar
Anwendungsprotokolle	Login möglich / Referenzdaten abrufbar / Rückgabewerte Testroutinen
`SNMP`	Abfrage beliebiger und zum Teil standardisierter Messwerte

1.3.2 Passive Überwachung

Abbildung 1.2: Passive Überwachung



Wie in Abbildung 1.2 zu erkennen, werden bei der passiven Überwachung nur Daten ausgewertet, welche durch Dienste selbst generiert werden oder aber durch eine Software, welche den Dienst lokal überwacht. Es erfolgt keine Abfrage bei den Diensten. Der zentrale Punkt, welcher auch die Auswertung übernimmt, bleibt passiv und empfängt lediglich Meldungen. Am häufigsten sind diese Meldungen LOG-Meldungen, generiert von einem LOG-System. Aber auch SNMP[4]-Traps[5] fallen unter diese Kategorie, daher auch die Wahl der Funktionsbezeichner in Abbildung 1.2.

[4]Simple Network Management Protocol: Ermöglicht eine plattformunabhängige Überwachung verschiedenster Endgeräte.
[5]SNMP-Trap: aktive Benachrichtigung des Monitors durch ein Endgerät/Dienst.

2 Logkorrelation in Cloud-Umgebungen

Im folgenden Kapitel wird eine Forschungsarbeit der Fachhochschule Fulda [5] vorgestellt. Ziel der Forschung war und ist es, ein gut skalierendes System zu entwickeln um eine automatisierte Auswertung von Syslog-Meldungen in Cloud-Umgebungen bereit zu stellen. Aufgrund der enormen Datenmengen die in solchen Umgebungen anfallen, kann eine Auswertung nur mittels korrelations- und Aggregationsverfahren geschehen. Um dieses Ziel zu erreichen, kommen verschiedene Standards und eine Reihe von Software-Lösungen zum Einsatz.
Nachfolgend werden einige wichtige Begriffe geklärt, die Anforderungen identifiziert, die verwendeten Standards und die eingesetzte Software erläutert und im weiteren Verlauf des Abschnitts wird anhand eines Beispiels eine Syslog-Korrelation vorgenommen.

2.1 Anforderungen

Viele Unternehmen haben in den letzten Jahren einen großen Teil ihrer IT-Infrastruktur ausgelagert. Laut Analysen werden bis 2025 80 % aller Unternehmen [6] weltweit ihre eigene Rechenzentrumsinfrastruktur abgeschaltet haben. Die wenigen Anbieter von Cloud-Lösungen stehen in Konkurrenz miteinander, daher existieren auch keine einheitlichen Schnittstellen um auf die Cloud-Konfigurationen zuzugreifen. Für die Überwachung, speziell von sicherheitskritischen Ereignissen, stehen ebenso nur proprietäre Schnittstellen jedes Anbieters zur Verfügung. Aus diesem Grund soll ein System geschaffen werden, das die gewaltige Menge an aufkommenden Logdaten, unabhängig vom Anbieter, analysiert und sicherheitskritische Ereignisse unverzüglich erkennt. Insbesondere soll das System das dynamisch sinkende und wachsende Syslog-Aufkommen beherrschen können. Denn durch die fluide Kostenstruktur der Cloud-Anbieter können schnell neue virtuelle Maschinen erstellt und entfernt werden, je nachdem wie viel Leistung der Kunde gerade benötigt. Darüber hinaus sollen Meldungen auch persistent gespeichert werden, vornehmlich zur Erstellung von Trends und Langzeitanalysen. Dabei soll der benötigte Speicherplatz so gering wie möglich gehalten werden.

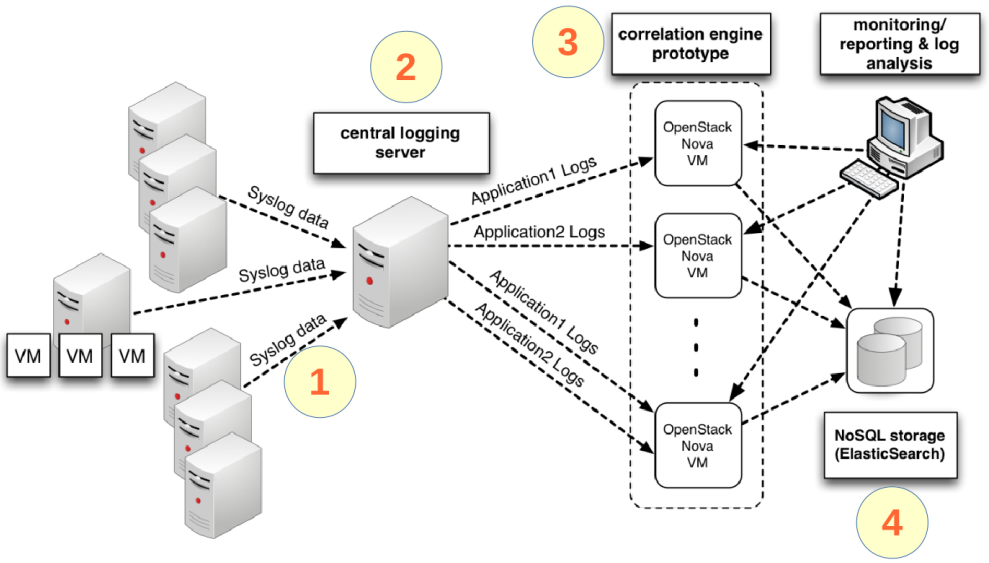
2.2 Beispielszenario

Im weiteren Verlauf dieses Kapitels soll zur detaillierteren Darstellung der Leistungsfähigkeit einer automatischen Syslog-Korrelation ein gängiges Angriffsszenario dienen: Eine SSH-BruteForce Attacke auf eine beliebige Anzahl der überwachten Systeme. Dabei soll genau der eine erfolgreiche Login innerhalb der enormen Anzahl an erfolglosen oder ungültigen Versuchen identifiziert werden. Dieses recht einfache Szenario ist bei der erheblichen Anzahl an möglichen Systemen (10K+) manuell unmöglich zu bewerkstelligen.

2.3 JCorrelat

In Abbildung 2.1 [5] ist der schematische Aufbau von JCorrelat dargestellt, ein Prototyp der die Anforderungen aus Abschnitt 2.1 erfüllen soll. Die hinzugefügten Nummern dienen der Übersichtlichkeit im weiteren Erklärungsverlauf.

Abbildung 2.1: Aufbau von JCorrelat



2.3.1 Syslog-Protokoll

Als Ereignisquelle und als Grundlage für neu generierte Meldungen wird das Syslog-Protokoll verwendet. Es ist das meist verbreitete Log-Format und wurde bereits in RFC 3164 standardisiert, jedoch sieht dieser Standard keine strukturierten Daten (`key` ⇒ `value`) innerhalb einer Syslog-Nachricht vor. Erst mit RFC 5424 [7] wurde diese Funktionalität hinzugefügt.

Tabelle 2.1: Aufbau RFC 5424

Feld	Inhalt	Beispiel
HEADER		
facility	*int* ∈ {0..23}	<**16**5> (`local0`)
severity	*int* ∈ {0..7}	<16**5**> (`Notice`)
timestamp	`RFC3339`	`2003-10-11T22:14:15.003Z`
hostname	string	`mymachine.example.com`
tag	string	`evntslog`
MESSAGE		
MSGID	string	`ID47`
structured data	key=value	`eventID="1011"`
content	string	`An application event log...`

Abbildung 2.2: Beispiel RFC5424 Syslog-Meldung

```
<165> 2003-10-11T22:14:15.003Z mymachine.example.com evntslog - ID47
[exampleSDID@32473 iut="3" eventSource="Application" eventID="1011"] BOMAn
application event log entry...
```

Tabelle 2.1 zeigt den Aufbau eines Syslog-Paketes nach RFC 5424 und Abbildung 2.2 die dazugehörige RFC 5424 konforme Nachricht. Die für die weitere Verwendung wichtigsten Felder wurden grün markiert. Dazu zählt der Schweregrad (*severity*), wobei 0 (`Emergency`) den höchsten und 7 (`DEBUG`) den niedrigsten Schweregrad darstellt. Außerdem das *tag*-Feld, das zum Beispiel die Herkunft (Programm) und die zugehörige *process ID* beinhalten kann. Am wichtigsten ist das *structured data*-Feld, welches mit beliebigen strukturierten Daten versehen werden darf.

2.3.2 Normalisierung von Syslog-Meldungen

Diese Sektion erläutert Punkt **2** in Abbildung 2.1. Es handelt sich um den zentralen Syslog-Server. Als Software kommt die Open Source - Lösung *rsyslog*[1] zum Einsatz. *rsyslog* ist RFC 5424 konform, äußerst performant und durch Module erweiterbar. Eines dieser Module kommt auch im hier vorgestellten Korrelationssystem zum Einsatz: *liblognorm*, mittlerweile ein fester Bestandteil von *rsyslog*.

Alle Applikationen senden ihre Syslog-Meldungen an diesen Server und damit auch alle SSH-Dienste. Somit passieren alle Syslog-Nachrichten über einen erfolgreichen, fehlgeschlagenen oder ungültigen Login dieses System. In Abbildung 2.3 wird jeweils ein Beispiel pro Fall abgebildet.

Abbildung 2.3: Beispiele SSH-Meldungen

```
Jan 29 16:00:25 HOST sshd[2039]: Accepted password for root from 10.0.23.4 port 39110 ssh2

Jan 29 16:06:00 HOST sshd[2032]: Failed password for root from 10.0.23.4 port 54548 ssh2

Jan 29 16:08:39 HOST sshd[2023]: Failed password for invalid user test from 10.0.23.4
port 57165 ssh2
```

liblognorm ist nun in der Lage diese Meldungen auf Basis vorher definierter Regeln zu durchsuchen (für das konkrete Beispiel finden sich die Regeln im Anhang unter Abbildung 5.2) und die relevanten Informationen zu extrahieren. Aus diesen Daten generiert *liblognorm* eine neue Syslog-Meldung in dem es aus den aufgeschlüsselten Feldern Protokoll, Nutzername, Port und Quelladresse strukturierte Daten bildet (Anhang: Abbildung 5.3). Zusätzlich wird die neue Meldung mit einem neuen *syslog-tag* names `SSHFAILURE` oder `SSHSUCCESS` versehen und an die Korrelationsinstanz weitergeleitet.

liblognorm normalisiert und serialisiert Daten, die aus verschiedenen Quellen stammen und unterschiedlich kodiert sein können zu neuen Nachrichten. Damit werden Redundanzen aus den Meldungen entfernt und eindeutige *syslog-tags* zur schnelleren Identifizierung durch die Korrelationsinstanz gesetzt.

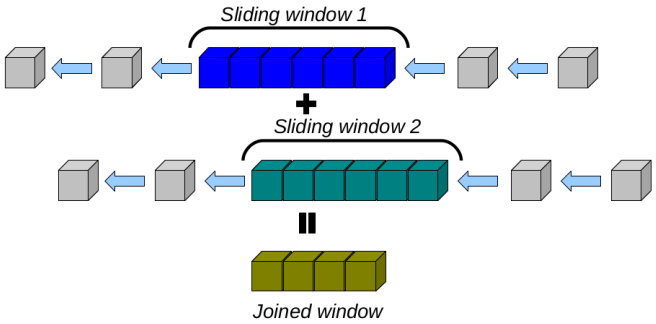
[1] https://rsyslog.com

2.3.3 Korrelation von Syslog-Meldungen

Mithilfe der normalisierten Meldungen wird nun versucht ein Zusammenhang zwischen den Syslog-Ereignissen zu erkennen. Diese Ereigniskorrelation ist der rechenaufwändigste Schritt und kann daher, wie in Punkt **3** in Abbildung 2.1 zu erkennen ist, auf mehrere virtuelle Systeme verteilt werden.

Die Korrelation der Syslog-Meldungen durch *JCorrelat* erfolgt mithilfe von *Drools-Fusion*[2]. *Drools-Fusion* ist eine *complex event processing engine*, mit dessen Hilfe zeitliche Schlussfolgerungen gezogen werden können. Um dieses Ziel zu erreichen werden Regeln erstellt um Events zu beschreiben und alle Nachrichten die zu einer Regel gehören werden analysiert, die restlichen Meldungen werden verworfen. Findet sich eine zeitliche Übereinstimmung, wird der Event ausgelöst, die Nachricht wird mit einem *syslog-tag* versehen und zur weiteren Analyse und Speicherung weitergeleitet. *Drools-Fusion* behält dazu alle relevanten Syslog-Meldungen in einer *in memory engine* vor und verwendet das Prinzip der *sliding windows* um Zusammenhänge zu erkennen. Abbildung 2.4 [8, Folie 70] demonstriert die Funktionsweise der *sliding-windows*, ein Quader ist im hier verwendeten Beispiel eine relevante Syslog-Meldung, die Fenster analysieren immer eine gewisse Anzahl an Meldungen aus verschiedenen Quellen, die in einem bestimmten Zeitfenster erzeugt wurden. Erfüllt das *joined-window* dann die Bedingungen in der definierten Regel, kann ein Event ausgelöst werden.

Abbildung 2.4: *sliding window*-Prinzip



Um nun, wie in Abschnitt 2.2 beschrieben, einen erfolgreichen SSH Brute-Force Angriff zu erkennen, muss zuerst eine Brute-Force-Attacke erkannt werden, dazu dient die Regel `SSH brute-force attempt` (Anhang: 5.4). Die Regel untersucht nur Syslog-Meldungen die mit dem *syslog-tag* `SSHFAILURE` versehen wurden. Es wird ein Event ausgelöst, wenn innerhalb von einer Minute vom gleichen Quell-Host und dessen verwendeten Benutzernamen zehn oder mehr fehlgeschlagene Login-Versuche erfolgen. Als Event wird in diesem Fall eine neue Syslog-Meldung verfasst und mit dem Schweregrad `WARNING` und dem *syslog-tag* `BRUTEFORCE` versehen. Die Meldung kann zum Beispiel persistent gespeichert werden (mehr dazu in Abschnitt 2.3.4) und an einen Monitor weitergeleitet werden.

Zur Erkennung einer erfolgreichen SSH Brute-Force-Attacke existiert ebenfalls eine beispielhafte Regel (Anhang: 5.5). Die Regel `Successful SSH brute-force attack` löst einen Event aus, wenn innerhalb von zehn Sekunden nach dem Auftreten von Meldungen mit dem *syslog-tag* `BRUTEFORCE` und `SSHFAILURE` und des zugehörigen Host und dessen verwendeten Benutzernamen, eine Meldung mit dem *syslog-tag* `SSHSUCCSESS` auftritt, welche vom gleichen Host und aus der dazugehörigen Menge der Benutzernamen stammt. Die generierte Syslog-Meldung wird mit dem gewichtigsten Schweregrad `EMERGENCY` und dem *syslog-tag* `INCIDENT` markiert.

Mit diesen beiden Regeln ist es *JCorrelat* unter der Zuhilfenahme von *Drools-Fusion* möglich den einen erfolgreichen Login unter tausenden Loginversuchen innerhalb einer Brute-Force-Attacke zu identifizieren.

[2] https://www.drools.org/

2.3.4 persistente Speicherung

In diesem Abschnitt wird Punkt **4** aus Abbildung 2.1 erläutert.

Neben der direkten Alarmierung von IT-Sicherheitskritischen Ereignissen, ist ein weiteres Ziel der IT-Sicherheit die Verfügbarkeit älterer Ereignisse, um auch über längere Zeiträume hinweg verbindlich Auskunft über den Zustand eines Systems zu geben oder die Daten mit neuen Methoden zu analysieren. Jedoch stellt die Speicherung von einer großen Menge sich ändernder Daten eine große Herausforderung dar. Mit *JCorrelat* ist es möglich eine Vielzahl an Diensten zu überwachen, dabei können sich die zu betrachtenden Daten ständig ändern und Dienste können hinzukommen oder wegfallen.

Ein erster Ansatz wäre die Speicherung der normalisierten Daten (Abschnitt 2.3.2) in einer relationalen Datenbank. Aufgrund der dynamischen Natur der Daten müsste für eine effiziente Verarbeitung das Schema einer solchen Datenbank stetig angepasst werden. Dieser Aufwand ist enorm und nicht Zielführend. Im Gegensatz zum relationalen Ansatz wird bei *NoSQL*-Datenbanken (Not only SQL) kein oder nur ein minimales Schema gespeichert. Somit erlaubt dieses Konzept zu jeder Zeit eine Änderung der Datenstruktur. Darüber hinaus lassen sich *NoSQL*-Datenbanken über viele Instanzen hinweg skalieren und liefern Daten sehr schnell aus. Allerdings mit dem Nachteil, dass nicht auf allen Systemen zeitgleich die aktuellsten Daten zur Verfügung stehen.

Grundsätzlich lassen sich drei Konzepte von *NoSQL*-Datenbanken identifizieren. Zuerst sind die einfachen ***key-value***-Implementierungen zu nennen. Diese Datenbanken sind mit Abstand die schnellsten, da die Werte lediglich als *BLOB* (Binary Large OBject) abgespeichert und somit nicht interpretiert werden. Damit sind die Werte nicht durchsuchbar, umso einfacher ist dafür die zugrunde liegende Programmierschnittstelle.

Die ***column-oriented data stores***-Lösungen speichern die Daten in Tabellen, ähnlich wie relationale-Lösungen. Jedoch werden die Zeilen einer Tabelle in sogenannte shards (Scherben) unterteilt und die Spalten in Spaltengruppen, jeweils Abhängig von ihrem Schlüssel. Damit ist das Tabellenschema beliebig erweiterbar, aber eine Volltextsuche bietet diese Lösung nicht.

Davon abweichend arbeiten die ***document-based data stores***. Sie speichern die Daten in *documents* ab, welche eine Menge an beliebigen Objekten mit einer unterschiedlichen Anzahl an Attributen beschreiben. Objekte dürfen jederzeit neu angelegt werden und die Art und Anzahl ihrer *key-value*-Paare ist nicht von Interesse. Als größter Vorteil ist bei diesem Konzept die Möglichkeit zur Volltextsuche zu nennen.

Aus den letztgenannten Gründen kommt bei *JCorrelat* *elasticsearch*[3] zum Einsatz, was die Konzepte eines ***document-based data stores*** umsetzt. *JCorrelat* sendet die normalisierten und korrelierten Syslog-Meldungen für eine persistente Speicherung direkt an *elasticsearch*. Monitoring- und Visualisierungssysteme können nun periodisch auf diese Daten zugreifen und kritische Ereignisse melden und visualisieren. Eine rudimentäre Visualisierung des SSH Brute-Force -Szenarios wurde in [5] veröffentlicht (Abbildung 2.5). Alternative Möglichkeiten zur Visualisierung werden in Abschnitt 3 erläutert.

Abbildung 2.5: Einfache Visualisierung durch JCorrelat

Severity	Facility	Message
emerge	security	Accepted password for root from 10.0.23.4 port 54548 ssh2 [bruteforce]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
warn	security	SSH brute-force attack for root from 10.0.23.4
info	auth	Connection closed by 10.0.23.4 [preauth]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2

[3] https://www.elastic.co/products/elasticsearch

2.3.5 Leistungsbetrachtung

Wie schon in Sektion 2.3.4 erwähnt, kostet das Korrelieren der Syslog-Meldungen am meisten Rechenzeit und ist daher der limitierende Faktor. Insbesondere stellt die Skalierbarkeit von *Drools-Fusion* ein Problem dar, da es auf einer *in memory engine* basiert, müssten mehrere Instanzen auf den gleichen Arbeitsspeicher zugreifen können (*distributed memory*). Jedoch existieren laut [5] dafür keine leistungsfähigen OpenSource-Lösungen, sodass eine Partitionierung nach Quellanwendung empfohlen wird. Somit ist nur eine *JCorrelat*-Instanz für eine Anwendung zuständig. Es besteht ein Zusammenhang zwischen benötigtem Arbeitsspeicher und der Größe des Betrachtungsfensters. Benötigt eine Regel ein längeres Betrachtungsfenster umso mehr Meldungen müssen im Arbeitsspeicher gehalten werden.
Untersucht wird auch die Geschwindigkeit von *rsyslog*, das ohne Normalisierung auf gängiger Hardware bei einer Syslog-Nachrichtengröße von 512 Byte über 200.000 Syslog-Meldungen untersuchen und damit die maximale Anzahl an Nachrichten auf einer 1 GBit-Netzwerkschnittstelle verarbeiten kann.

Für den Benchmark wurde mithilfe der Software *loggen*[4] eine 20.000 Meldungen umfassende Datei in Dauerschleife eingelesen und die daraus generierten Syslog-Meldungen an den Syslog-Server (Punkt **2** in Abbildung 2.1) weitergeleitet. Die Datei bestand zu circa einem Drittel aus SSH-Meldungen, der Rest waren andere übliche Syslog-Meldungen ohne Relevanz für die Korrelation.

Abbildung 2.6: Ergebnisse des Benchmark

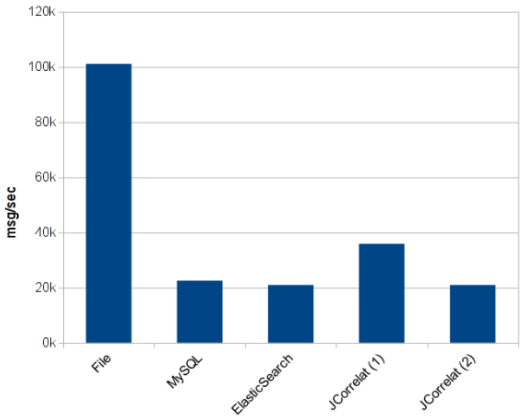


Abbildung 2.6 ([5]) zeigt die Ergebnisse eines Benchmarks über das gesamte Korrelationssystem hinweg.

File zeigt die Anzahl der Meldungen die normalisiert werden können, wenn die normalisierten Syslog-Meldungen lokal in Dateien geschrieben werden.

Die Balken **Mysql** und **ElasticSearch** demonstrieren die Anzahl der normalisierten Syslog-Meldungen, wenn diese direkt über *rsyslog*-eigene Module in diese Datenbanken geschrieben werden. Dabei wird eine auf HTTP-basierende Schnittstelle verwendet.

Über 30.000 normalisierte Syslog-Meldungen (**JCorrelat (1)**) können hingegen mit *JCorrelat* über die *elasticsearch*-API direkt in *elasticsearch* geschrieben werden, was die Differenz zum vorhergehenden Balken erklärt, da hier der HTTP-Verwaltungsaufwand entfällt.

Bei **JCorrelat (2)** wurde die Korrelationsfunktion aktiviert. Damit ist *JCorrelat* genau so performant, als wenn die normalisierten Daten lediglich in eine Datenbank geschrieben würden.

[4] https://github.com/balabit/syslog-ng/blob/master/tests/loggen/loggen.md

3 Auswertung und Visualisierung

Da *JCorrelat* wohlformatierte Daten in *elasticsearch* ablegt, ist es möglich mit einer Vielzahl an weiteren Werkzeugen diese Daten zu analysieren und zu visualisieren. Aus diesem Grund wurde auf der Basis von [9] ein ELK-Stack aufgebaut um weitere beispielhafte Visualisierungen vorzustellen. Der Aufbau des verwendeten ELK-Stacks ist Schematisch in Abbildung 5.6 (Anhang) dargestellt. Für die Visualisierung zeigt sich die *Kibana* verantwortlich.
Abbildung 3.1 zeigt eine durch *Kibana* erstellte Visualisierung. Dabei greift *Kibana* auf Daten zurück, die durch *Logstash* strukturiert in *elasticsearch* abgelegt wurden. Ähnlich wie im Beispielszenario, (Abschnitt 2.2) sind auch hier fehlgeschlagene SSH-Logins veranschaulicht. Die IP-Quelladressen wurden dabei mithilfe von *Geotargeting* einer möglichen Region zugewiesen, ebenso die Anzahl der erfolglosen Loginversuche aus dieser Region über einen vorher definierten Zeitraum. Der Kreisdurchmesser hat keinerlei Bedeutung, lediglich der Farbverlauf gibt in fünf dynamisch errechneten Abstufungen die Anzahl der erfolglosen Versuche an, wie man am Beispiel Kiev deutlich sieht. Um auch alle erfolgreichen Logins anzuzeigen ist eine weitere Karte notwendig (Abbildung 3.2).
Diese Form der Illustration würde sich auch für *JCorrelat* anbieten, dann könnten die Ereignisse (Brute-Force-Attacke und erfolgreicher Login während einer Attacke) auf einer Karte darstellt werden.

Abbildung 3.1: Visualisierung mit Kibana (fehlgeschlagene Logins)

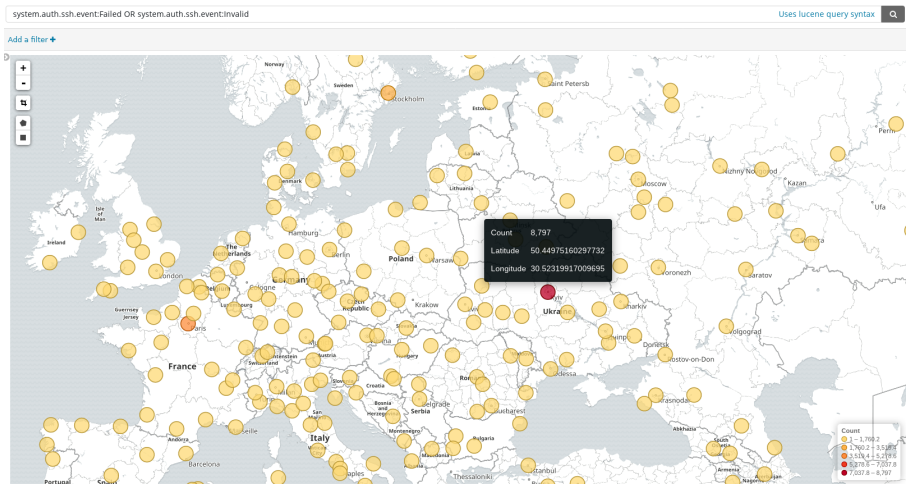
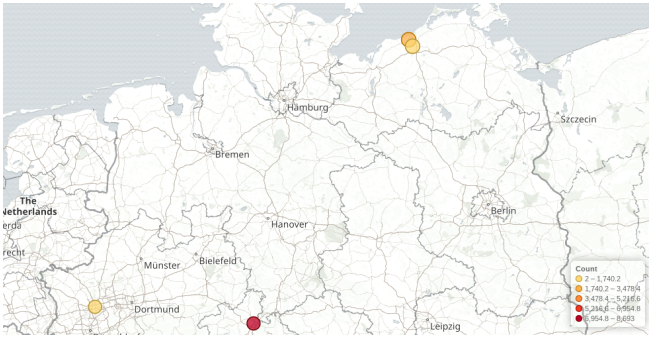


Abbildung 3.2: Visualisierung mit Kibana (erfolgreiche Logins)



4 Fazit und Ausblick

Mit *JCorrelat* wurde ein Prototyp für die automatische Korrelation und Konsolidierung von Syslog-Meldungen aus unterschiedlichen Quellen geschaffen. Wie die Ergebnisse des Benchmark zeigen, skaliert der Prototyp sehr gut, wenn für jede Anwendung eine *JCorrelat*-Instanz verwendet wird. Weniger Verwaltungsaufwand würde allerdings die Verwendung einer (eventuell auch proprietären) *distributet-memory*-Lösung bieten, da dann die *Drools-Fusion*-Regeln global platziert werden könnten und jede *Drools*-Instanz auf den selben Daten operieren könnte. Arbeitsspeicher ist der limitierende Faktor des Korrelationssystems. Je mehr Arbeitsspeicher zur Verfügung steht umso mehr und größere Betrachtungsfenster kann *Drools-Fusion* verwenden und so bessere Ergebnisse erzielen.
Wie fortlaufend am Beispiel aus Sektion 2.2 gezeigt, ist der Prototyp in der Lage sicherheitskritische Ereignisse sicher zu identifizieren und eignet sich so bestens für die Überwachung von großen IT-Umgebungen. Anzumerken ist allerdings, dass die Zeitfenster in den Beispielregeln (Anhang: Abbildung 5.4 und Abbildung 5.5) recht willkürlich gewählt wurden und in Produktivumgebungen sicherlich angepasst werden müssten.

Durch die Normalisierung der Syslog-Meldung wird eine sehr effiziente Verdichtung von Informationen erreicht und Redundanzen werden größtenteils eliminiert. Das führt neben der damit verbundenen hohen Ausführungsgeschwindigkeit der Korrelation auch zu einem stark reduzierten Speicherbedarf für die persistente Speicherung. Durch die Wahl, die normalisierten und korrelierten Daten in *elasticsearch* abzuspeichern, bieten sich vielfältige Möglichkeiten der Auswertung und Visualisierung an. So könnten bereits bestehende Monitoringsysteme um Module erweitert werden, welche die zuvor erstellten Ereignisse periodisch abfragen und somit alarmieren können.

Auch eine Korrelation von weiteren Daten ist realisierbar, so könnten die Syslog-Meldungen zum Beispiel mit Geo- oder Performance-Informationen angereichert werden. Falls nicht genügend Arbeitsspeicher zur Verfügung steht, aber eine hohe Korrelationsgenauigkeit erreicht werden soll und die Korrelationsgeschwindigkeit keine Rolle spielt, könnte *JCorrelat* auch auf den Datenbestand in *elasticsearch* zugreifen. Um nicht alle Regeln (je Dienst) einzeln konfigurieren zu müssen, soll das Ziel sein, *JCorrelat* mit Hilfe von Vorlagen konfigurieren zu können. Damit könnten neu hinzukommende Dienste einer automatischen Korrelation unterzogen werden. Darüber hinaus wäre auch eine Integration in *kubernetes*[1], *openstack*[2] oder dem ELK-Stack (Abschnitt 3) erstrebenswert.

[1] https://kubernetes.io/
[2] https://www.openstack.org/

Literaturverzeichnis

[1] Guillermo Francia, Levent Ertaul, Luis Hernandez Encinas, and Eman El-Sheikh. *Computer and Network Security Essentials*. Springer, 2017.

[2] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Die Lage der IT-Sicherheit in Deutschland 2017*.

[3] Bundesamt für Sicherheit in der Informationstechnik (BSI). https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/Aktivitaeten/Cyber-Abwehrzentrum/cyberabwehrzentrum_node.html. Abrufdatum: 03.06.2018.

[4] Bundesamt für Sicherheit in der Informationstechnik (BSI). https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/Aktivitaeten/IT-Lagezentrum/itlagezentrum_node.html. Abrufdatum: 03.06.2018.

[5] Sven Reissmann, Dustin Frisch, Christian Pape, and Sebastian Rieger. Correlation and consolidation of distributed logging data in enterprise clouds. *International Journal on Advances in Internet Technology Volume 7, Number 1 & 2, 2014*, 2014.

[6] IX Magazin für professionelle Informationstechnik. https://www.heise.de/ix/meldung/Umfrage-Cloud-kannt-teuer-werden-4072912.html. Abrufdatum: 08.06.2018.

[7] R. Gerhards. https://tools.ietf.org/html/rfc5424. 2009.

[8] T. Surdilovic. https://slideshare.net/tsurdilovic/jboss-drools-and-drools-fusion-cep-making-business-rules-react-to-rte. 2011.

[9] Patrick Kleindienst. Building a real-world logging infrastructure with logstash, elasticsearch and kibana. 2016.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

Abbildung 5.1: Statusverlauf visualisiert mit *NagVis*

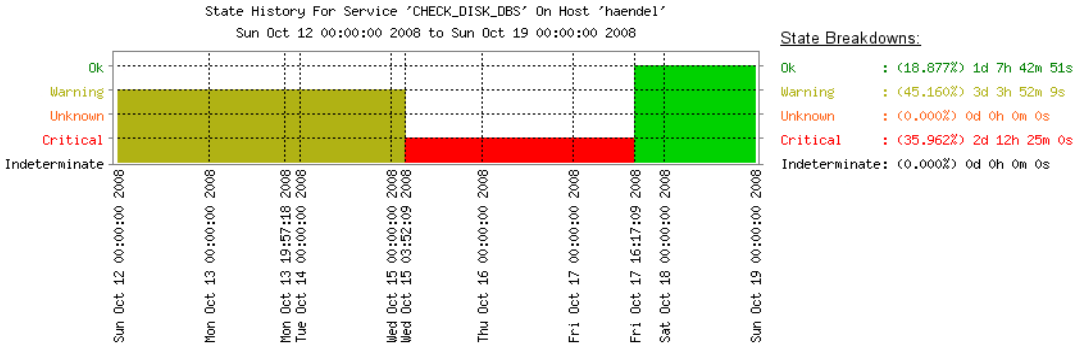


Abbildung 5.2: *liblognorm* - Regel

```
rule=SSHSUCCESS : Accepted password for %user:
word% from %ip:ipv4% port %port : number% %protocol:word%

rule=SSHFAILURE : Failed password for %user:
word% from %ip:ipv4% port %port : number% %protocol:word%

rule=SSHFAILURE : Failed password for invalid user %user:
word% from %ip:ipv4% port %port : number% %protocol:word%
```

Abbildung 5.3: *liblognorm* - strukturierte Daten

```
{   "data": {
        "protocol":"ssh2",
        "port" : "54548",
        "ip": "10.0.23.4",
        "user": "root"
    },
    "time":"2014-01-29T16:06:00.000",
    "host":"test.example.com",
    "facility":"auth",
    "severity":"info",
    "program" :"sshd",
    "message":" Failed password for root from
                10.0.23.4 port 54548 ssh2",
    "tags" : ["SSHFAILURE"] }
```

Abbildung 5.4: *Drools* Brute Force - Regel

```
        rule "SSH brute-force attempt"
        no-loop
        when
            Message (   $host:host,
                        $user:data["user"])
            $atts: CopyOnWriteArrayList(size >= 10)
                from collect(
                    Message(    tags contains"SSHFAILURE",
                                host == $host,
                                data["user" ] == $user)
                    over window : time (1m))
        then
            Message last = (Message) $atts.get($atts.size( )-1) ;

            for (Object f: $atts) {
                retract ( f ) ;
        }

            insert (messageFactory(last)
                . setTime(last.getTime( ))
                . setSeverity(Message.Severity.WARNING)
                . setFacility(Message.Facility.SECURITY)
                . setMessage("SSH brute-force attack" +
                    "for @{data.user} from @{data.ip}")
                . addTag ("BRUTEFORCE")
                . message( )) ;
        end
```

Abbildung 5.5: *Drools* - login detection Regel

```
rule "Successful SSH brute-force attack"
no-loop
when
     $ att: Message (    tags contains "SSHFAILURE",
                         tags contains "BRUTEFORCE",
                         $host: host ,
                         $user: data ["user"])
     $ suc: Message (    host == $host,
                         data ["user" ] == $user,
                         tags contains "SSHSUCCESS",
                         this finishes[10 s] $att)
then
     $att.addTag("INCIDENT");
     $att.setSeverity(Severity.EMERGENCY);
     $att.setMessage($att.getMessage( ) + "[bruteforce]");
update ($att);
end
```

Abbildung 5.6: Aufbau des verwendeten ELK-Stacks