INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Erkennung von Innentätern – Entwicklung eines Systems zur heuristischen Analyse von Logfiles am Beispiel eines Hochschulrechenzentrums

Michael Grabatin

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Erkennung von Innentätern – Entwicklung eines Systems zur heuristischen Analyse von Logfiles am Beispiel eines Hochschulrechenzentrums

Michael Grabatin

Aufgabensteller:	Priv. Doz. Dr. H. Reiser
Betreuer:	Felix von Eye Dr. Wolfgang Hommel Stefan Metzger
Abgabetermin:	2. August 2012

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 2. August 2012

.....

(Unterschrift des Kandidaten)

Abstract

Die Erkennung von Angriffen durch Innentäter und die Identifizierung kompromittierter Benutzerkennungen ist keine triviale Aufgabe.

Das in dieser Arbeit weiterentwickelte Programm `LoginIDS` untersucht Logdateien von verschiedenen System-Diensten auf Anomalien im Benutzerverhalten mit dem Ziel, Innentäter oder kompromittierte Accounts zu erkennen. Als neues Quell-Format wurde das Logformat des Apache-Webservers zu den Formaten der SSH-Server-Logs und Citrix-Windows-Terminalserver-Logs hinzugefügt. Alle Logdateien werden durch `LoginIDS` mit einem heuristischen Verfahren auf Anomalien im Nutzerverhalten hin überprüft. Zu dieser Analyse wird eine Datenbank verwendet, in der die Logeinträge zu Statistiken zusammengefasst werden. Dabei wird auch das geltende Datenschutzrecht beachtet und geeignete Retentionsintervalle verwendet. Die gefundenen Auffälligkeiten werden als Alarme gemeldet. Diese können in Trigger-Skripten behandelt und in dem Logfile-Management-System `Splunk` angezeigt werden.

Es wurde festgestellt, dass `LoginIDS` als eine Komponente in einem Sicherheitssystem eingesetzt werden kann, um die manuell näher zu untersuchenden Log-Einträge deutlich zu reduzieren. `LoginIDS` ist unter der GPLv3 veröffentlicht.

Inhaltsverzeichnis

1. Einleitung

Das erste Kapitel gibt einen Überblick über diese Arbeit, das Umfeld in dem sie entstand sowie eine Zusammenfassung der Aufgabenstellung. Da das weiter zu entwickelnde System, genannt `LoginIDS`, zu Beginn der Arbeit schon als Prototyp existierte, wird außerdem auf die bereits vorhandene Funktionalität eingegangen. Zum Schluss wird die Struktur der weiteren Arbeit erläutert.

1.1. Motivation

Beim Betrieb eines Netzes, in dem verschiedenste Dienste angeboten werden sollen, fallen große Mengen von Log-Daten an. Das Durchsuchen dieser Logs zum Auffinden sicherheitsrelevanter Vorfälle von Hand, ist sehr mühsam und arbeitsintensiv. Um diese Arbeit zu vereinfachen und die Menge der zu untersuchenden Daten zu verringern, werden Programme eingesetzt, die anhand unterschiedlicher Kriterien versuchen, die wichtigen Meldungen von den unwichtigen zu trennen.

Zur Erkennung und Abwehr von externen Angriffen gibt es bereits viele Lösungen. Darunter fallen zum Beispiel Firewalls, Virenscanner oder Intrusion Detection Systeme. Der Fokus dieser Arbeit liegt hingegen auf der Erkennung von Bedrohungen durch Innentäter. Daher soll zunächst geklärt werden, wie hier der Begriff Innentäter oder Insider-Threat, wie er in der englischsprachigen Literatur verwendet wird, definiert ist. Dabei halte ich mich an die folgende Definition, die auch in der ersten Arbeit zu `LoginIDS` verwendet wurde:

„Unter Insider-Threat werden alle Aktivitäten zusammengefasst, die unter missbräuchlicher Verwendung von Wissen, Zugangsberechtigungen sowie der aktiven Ausnutzung von Schwachstellen eine Gefährdung der Sicherheit eines Unternehmens, insbesondere bezüglich der Vertraulichkeit, Integrität und Verfügbarkeit von Informationen und IT-Systemen darstellen und sowohl materiellen Schaden als auch negative Auswirkungen auf die Reputation einer Organisation zur Folge haben können.“ [vEMH12b]

Die Notwendigkeit, den eigenen Angestellten oder anderweitig Zugangsberechtigten nicht bedingungslos zu vertrauen, zeigt das Ergebnis einer britischen Studie [NS12]. Dort wurde ermittelt, dass die Hälfte der Angestellten ihre Firmen-Passwörter für weniger als fünf Pfund (entspricht etwa sechs Euro, Stand April 2012) verkaufen würden. Der geringe Wert der Sicherheit des eigenen Unternehmens für Angestellte wird durch eine Analyse von [BHH+11] unterstützt. Dort wird darauf hingewiesen, dass im vorhergehenden Bericht 48%, im aktuellen nur noch 17%, der gemeldeten Angriffe von innen durchgeführt wurden. Dieser große Rückgang wird nach Aussagen der Autoren nicht durch eine geringere Anzahl an Innentätern,

sondern durch eine gestiegene Zahl von externen Angriffen erklärt. Zudem zeigt diese Studie, dass 85% der Angriffe von innen durch normale Angestellte oder Endkunden begangen werden. Angestellte im Management tragen nur mit 11% bei, System- und Netzwerkadministratoren sowie Softwareentwickler bleiben unter 5%. Der Schaden, den ein Mitarbeiter mit weitreichenden Berechtigungen, wie zum Beispiel ein Manager, anrichten kann, ist aber auch größer.

Um diese Statistiken etwas besser einordnen zu können, wird die Umgebung, in der diese Arbeit geschrieben und das entwickelte System `LoginIDS` getestet werden, im Folgenden näher betrachtet. Das Leibniz-Rechenzentrum betreibt unter anderem für die Münchner Hochschulen und die Bayerische Akademie der Wissenschaften verschiedenste IT-Dienste. Nach Angaben des Jahresberichtes von 2011 [Lei11], werden dabei E-Mail-Server für 116.077 Studenten und Mitarbeiter betrieben. Davon sind 93.654 Kennungen, über die E-Mail, WLAN und ein VPN-Zugang verwendet werden können, an Studenten vergeben. Zu den erbrachten Diensten gehört auch der Betrieb der Kommunikationsinfrastruktur, der Unterhalt einer großen Datenarchivierungsinfrastruktur sowie der Betrieb von Hoch- und Höchstleistungsrechnern.

Ein großes Problem ist hier die hohe Fluktuation von Mitarbeitern und Nutzern. Vor allem studentische Hilfskräfte werden oft nur im Rahmen von studentischen Arbeiten oder Praktika für ein Semester angestellt. In dieser Situation ist es schwierig, den Überblick über Zugriffe auf die vielen angebotenen Dienste zu behalten.

So ist es beim Schutz prinzipiell jeder Organisation nicht ausreichend, nur auf Firewalls und Intrusion Detection Systeme zu vertrauen, die primär zur Abwehr externer Angriffe gedacht sind. Es sollte auch ein System geben, das die erfolgreichen Zugriffe auf Konformität mit dem üblichen Verhalten eines Benutzers vergleicht und Hinweise auf unberechtigte Zugriffsversuche meldet. Das Tool `LoginIDS` soll genau diese Analyse durchführen. Dabei kann `LoginIDS` nicht direkt als klassisches Host- oder Netzbasiertes Intrusion Detection System eingeordnet werden. Die Analyse der Logfiles erfolgt auf einem zentralen System, aber die Logfiles müssen nicht unbedingt von diesem System stammen. Stattdessen können sie von verschiedenen Systemen im Netzwerk gesammelt werden. Die Analyse der Logeinträge erfolgt dabei nach einem heuristischen Verfahren. Das heißt, `LoginIDS` lernt das normale Login-Verhalten von Nutzern und meldet Abweichungen als potentielle Angriffe.

Die Notwendigkeit einer automatisierten Überprüfung zeigt die folgende Abbildung 1.1. Sie gibt einen Überblick über die Menge der Logeinträge, die auf einem `SSH`-Gateway und einem Citrix-Windows-Terminalserver, im Folgenden meist nur als Windows-Terminalserver bezeichnet, generiert werden. Es ist gut zu erkennen, dass an Wochenenden kaum Zugriffe stattfinden, unter der Woche aber, über alle Services, regelmäßig um die 400 Logeinträge pro Tag erzeugt werden. Selbst wenn ein Mitarbeiter die Logdateien von Hand nach Auffälligkeiten durchsuchen würde, ist die Wahrscheinlichkeit groß, dass bei dieser sehr monotonen und langwierigen Arbeit relevante Ereignisse übersehen werden.

Um die Menge, der zu überprüfenden Logeinträge, signifikant zu verringern, ermittelt das Tool `LoginIDS` das für jeden Benutzer übliche Verhalten. Weicht der Benutzer nach einer Lernphase von dem ermittelten Normalverhalten ab, markiert das System diesen Vorfall.

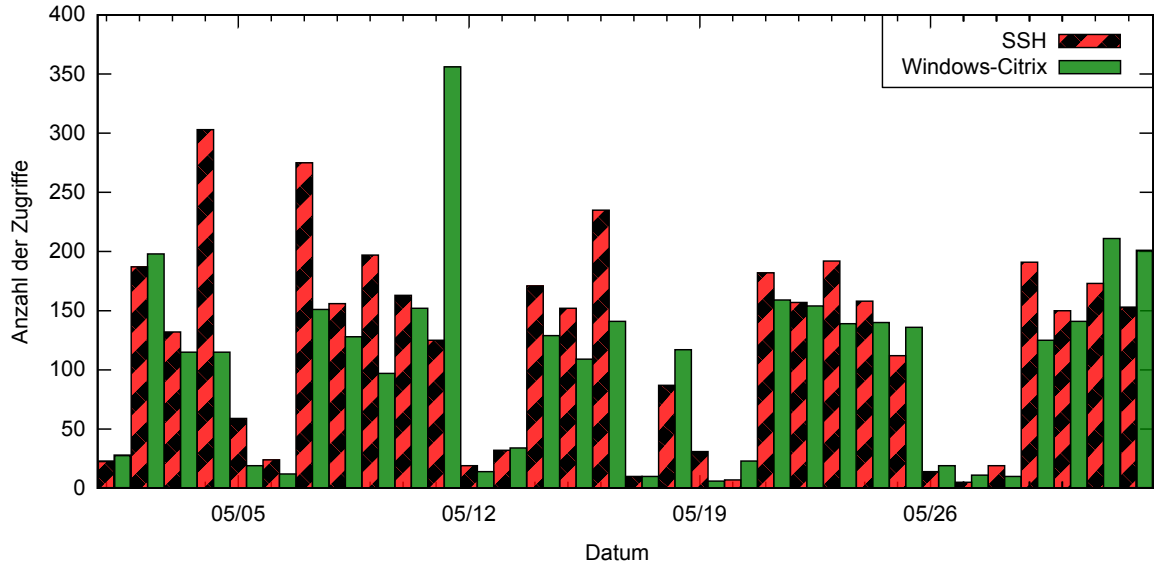

Abbildung 1.1.: Übersicht über die Anzahl der Logeinträge auf dem Testsystem

Ob dieser Vorfall nun tatsächlich durch einen kompromittierten Account, einen Sicherheitsverstoß oder doch durch legitimes Verhalten zu erklären ist, ist nach wie vor die Aufgabe desjenigen, der die Ausgabe analysiert. Es geht hierbei darum, die Menge der zu überprüfenden Logeinträge signifikant zu verringern.

1.2. Aufgabenstellung und Ziele

In dieser Arbeit soll das als Prototyp schon bestehende `LoginIDS` zuerst konzeptionell erweitert werden. Anschließend werden die Konzepte unter der Verwendung der Programmiersprache Perl implementiert. Die Erweiterungen beziehen sich besonders auf die im Folgenden genannten Punkte.

Ein Ziel ist es, dass neben `SSH` und Windows-Terminalserver-Logins auch Apache-Webserver-Logins mit `mod_auth` untersucht werden können. Hierzu wird ein Skript entwickelt, das entsprechende Logfiles konvertieren kann. Dieses System von Konvertierungsskripten sollte möglichst einfach an weitere Dienste angepasst werden können.

Ein besonders wichtiger Punkt in der Weiterentwicklung des Projektes ist, eine geeignete Visualisierung für die Ausgabe zu erstellen. Um dies umzusetzen, wird eine Integration der `LoginIDS`-Ausgabe in ein Security Information & Event Management System (SIEM) angestrebt. Die Wahl der SIEM-Lösung fällt dabei auf das Produkt „`Splunk`“.

Außerdem soll der Algorithmus, mit dem verdächtige Logins erkannt werden, so überarbeitet werden, dass veraltete Einträge, entsprechend des jeweiligen Anwendungsfalls, nach einem einstellbaren Zeitintervall gelöscht werden.

Zudem werden im Anschluss an dieses Kapitel noch verschiedene andere technische und wissenschaftliche Ansätze zur Vorhersage von Insider Threats und deren Eignung für eine Umsetzung am LRZ untersucht.

1.3. Stand von `LoginIDS` vor Beginn der Arbeit

Zu Beginn dieser Arbeit ist ein Prototyp von `LoginIDS` schon einige Monate im Testbetrieb. Genauer beschrieben wurden die Funktionen in [HMRvE12, Abschnitt 4.1] und in [vEMH12b, Abschnitt 3]. Im Vergleich mit anderen Sicherheitslösungen in Kapitel 2 wird `LoginIDS` als ein heuristisches System klassifiziert. Hier wird im Folgenden nur ein einleitender Überblick über die grundlegenden Funktionalitäten des Programms gegeben. Eine detailliertere Beschreibung findet sich in Kapitel 3.1. Dort werden sowohl bestehende als auch neue Anforderungen und geplante Änderungen an dem bestehenden System vorgestellt.

Einmal täglich werden `SSH`-Server-Logs und Citrix-Windows-Terminalserver-Logs über einen Cronjob gesammelt und `LoginIDS` auf ihnen ausgeführt. `LoginIDS` besteht dabei aus mehreren Perl-Skripten und einem Shell-Skript. Das Shell-Skript wird über einen Cronjob gesteuert, einmal täglich ausgeführt und koordiniert die Arbeit der anderen Programmteile. So werden Logfiles, die in einem Eingabeordner abgelegt wurden, an die jeweiligen Konvertierungsprogramme übergeben. Diese Perl-Programme wandeln das spezielle Logfileformat eines Dienstes in ein universelles Format, das dann vom Hauptprogramm verarbeitet werden kann. Die Ausgabe des Hauptprogramms, welches mit Hilfe einer Datenbank Muster im Benutzerverhalten erkennt, wird wiederum in ein Logfile geschrieben. Die Abbildung 1.2 zeigt diesen Ablauf schematisch.

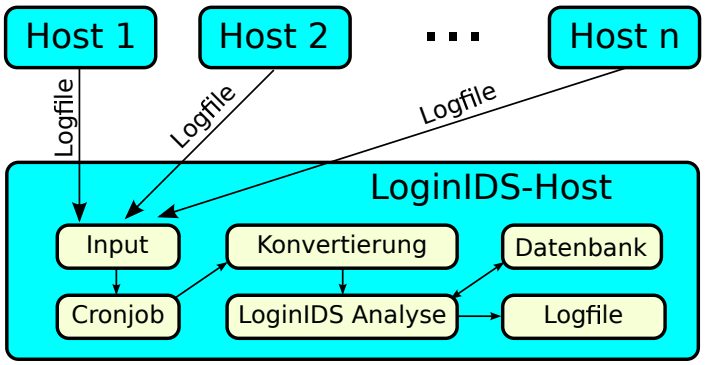

Abbildung 1.2.: Übersicht über den prinzipiellen Ablauf des Prototypen

Mit Hilfe des Logfiles können die verdächtigen Logins untersucht werden. Zusätzlich können Informationen aus der Datenbank verwendet werden, um die Beziehungen einzelner Ereignisse zueinander leichter herzustellen. Dort werden in dem sogenannten `DetailedLog` alle Ereignisse nochmal mit wichtigen Zusatzinformationen wie Benutzername, Quell- und Zieladresse und einem Zeitstempel festgehalten.

Die ausgegebenen Alarmmeldungen unterscheiden zwischen `NEW USER`, `FIRST LOGIN` und `SUSPICIOUS LOGIN`. `NEW USER` bezeichnet dabei eine Anmeldung mit einer `LoginIDS` noch nicht bekannten Kennung. Der Alarm `FIRST LOGIN` wird bei einem neuen, für einen Benutzer bislang unbekannten Tupel Quell-, Zielhost gemeldet. Ein `SUSPICIOUS LOGIN` wird ausgegeben, wenn die beobachtete Anmeldung zwar prinzipiell bekannt ist, aber zum ersten Mal zu dieser Zeit beobachtet wurde. Die Zeit wird dabei durch den Wochentag und die Stunde des Zugriffes festgelegt.

Aus Datenschutzgründen ist vorgesehen, Einträge im `DetailedLog` nach sieben Tagen zu löschen. Die Analyse von Loginvorgängen läuft über das `LongtimeLog`, eine separate Tabelle, die weniger personenbezogene Daten enthält, beziehungsweise diese über einen längeren Zeitraum aggregiert und mittelt.

1.4. Struktur der Arbeit

In Kapitel 2 werden Systeme betrachtet, die vergleichbare Aufgaben, wie das Tool `LoginIDS`, erfüllen. Außerdem wird der BSI-Grundschutzkatalog auf nützliche Hinweise im Umgang mit Innentätern hin untersucht. In Kapitel 3 wird zuerst eine detaillierte Beschreibung des `LoginIDS`-Prototypen gegeben, danach werden die Anforderungen an `LoginIDS` konzeptionell untersucht. Die sich daraus ergebenden neuen Funktionen werden in Kapitel 4 näher betrachtet. Funktionen, die im Rahmen dieser Arbeit umsetzbar sind, werden dann implementiert und in Kapitel 5 dokumentiert. Kapitel 6 beschäftigt sich mit der Umsetzung der Visualisierung mit Hilfe der Logfile-Management-Funktionen in `Splunk`. In Kapitel 7 werden Tests und deren Ergebnisse geschildert. Das letzte Kapitel 8 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf Problemstellungen, die nicht in dieser Arbeit bearbeitet wurden, in weiteren Arbeiten aber untersucht, beziehungsweise implementiert werden können.

2. Ansätze zur Vorhersage von Insider Threats

In diesem Kapitel werden momentan häufig eingesetzte Systeme zur Angriffserkennung und Abwehr betrachtet. Außerdem wird untersucht wie mit dem Thema „Innentäter“ in anderen Arbeiten und Veröffentlichungen umgegangen wird.

2.1. Allgemeine Ansätze und Techniken

Die Idee, verdächtige Ereignisse im Netzwerk automatisiert auszuwerten und Benachrichtigungen zu generieren, ist nicht neu. In diesem Abschnitt wird die Funktionalität von `LoginIDS` von schon bestehenden Programmen und Tools mit ähnlicher Funktionalität abgegrenzt.

2.1.1. Honeypots

Honeypots werden in verschiedenen Quellen oftmals unterschiedlich definiert. Durch die große Anzahl an unterschiedlichen Anwendungsgebieten ist das nicht verwunderlich. Eine allgemeine Definition ist: *„A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.“* [Spi03] Honeypots sind also das digitale Äquivalent zu dem gezielten Verteilen von (Falsch)-Informationen, um herauszufinden, welche Mitarbeiter diese weitergeben oder anderweitig verwenden. Ähnlich dem Ziel von `LoginIDS`, den Aufwand zum Erkennen von Angriffen möglichst weit zu reduzieren, senkt ein Honeypot ebenfalls drastisch die Anzahl der zu überprüfenden Logeinträge. In der Regel gibt es für einen Benutzer keinen legitimen Grund, überhaupt auf einen Honeypot zuzugreifen. Daher kann jeder erkannte Zugriffsversuch einem Angriff gleichgesetzt werden. Hierdurch können potentielle Angreifer, die das Netz nach lohnenden Informationen oder Diensten mit Schwachstellen durchsuchen, angelockt und identifiziert werden. Diese Angreifer müssen nicht unbedingt menschlich sein, auch automatisierte Angriffstools oder Würmer fallen auf solche Systeme rein. Unter Umständen sollte aber berücksichtigt werden, dass auch zufällige Zugriffe, zum Beispiel durch Tippfehler bei der Eingabe von IP-Adressen, passieren können. Daher müssen auch hier Alarmmeldungen genauer analysiert und bewertet werden.

Honeypots haben allerdings einen Nachteil, wenn es um die Identifizierung von Innentätern geht. Entsprechend ihrer Position im Unternehmen wissen diese unter Umständen von einem entsprechenden System und können es dadurch vermeiden, durch einen Zugriff aufzufallen.

2.1.2. Signaturbasierte Intrusion Prevention Systeme

Mit einem Honeypot kann der Zugriff auf ein bestimmtes in der Regel nicht produktiv eingesetztes System überwacht werden. Um Produktivsysteme zu überwachen, kommen in nahezu allen größeren Organisationen, wie auch dem LRZ, Intrusion Prevention Systeme zum Einsatz. Intrusion Prevention Systeme analysieren meist nahezu in Echtzeit Logfiles oder Netzwerkverkehr, um ungewöhnliche Ereignisse zu erkennen und eventuell sofort Gegenmaßnahmen zu ergreifen. Diese Systeme arbeiten mit Regelsätzen und versuchen Angriffssignaturen zu erkennen.

Ein Beispiel für ein solches System ist `Fail2Ban`, welches unter anderem dazu verwendet werden kann `SSH`-Brute-Force-Angriffe oder Port-Scans zu erkennen und sofort Firewallregeln zu erstellen, die den Angreifer aussperren. Da in einem solchen System prinzipiell Logfiles von allen Systemen, entweder lokal oder auf einem dedizierten Host, überprüft werden können, ist es selbst mit dem Wissen von der Existenz eines solchen Systems schwer, nicht aufzufallen. Mit viel Zeit und einem langsamen Vorgehen, um keine Schwellwerte zu überschreiten, lassen sich diese Systeme aber auch überwinden.

Vollständig ignoriert werden aber normalerweise erfolgreiche Logins. Außerdem wird nicht darüber Buch geführt, welcher Nutzer schon einmal ein bestimmtes System verwendet hat.

2.1.3. Heuristische Systeme

Um Anomalien im Vergleich zu normalem Netzwerkverkehr zu entdecken, werden heuristische Systeme eingesetzt. Das Prinzip von heuristischen Systemen ist, dass sie durch die Beobachtung eines Normalverhaltens Anomalien aus dem Netzwerkverkehr, ohne die Angabe von konkreten Regelsätzen, erkennen. Ein solches System verwendet Methoden des Data Mining, um aus einem Normalverhalten selber Regeln zu erstellen, die im Betrieb mit dem aktuellen Netzwerkverkehr verglichen werden. Dieser Abgleich der aktuellen Daten mit den in der Vergangenheit gesammelten Daten kann entweder manuell oder automatisch erfolgen.

In dieser Klassifizierung von Systemen zum Erkennen von Angreifern, lässt sich `LoginIDS` als heuristisches System einordnen. Anstatt Abweichungen im Netzwerkverkehr zu erkennen, spezialisiert sich das Programm auf Loginvorgänge für verschiedene Dienste.

Zusätzlich zu den oben genannten technischen Maßnahmen gibt es auch organisatorische Ansätze, die zur Vermeidung oder Erkennung von Angriffen durch Innentätern hilfreich sein können. Der nächste Abschnitt 2.2 beschreibt ein komplexes System, welches sowohl organisatorische, technische als auch psychologische Ansätze verwendet. Der Abschnitt 2.3 beschäftigt sich mit einer Sammlung von organisatorischen und technischen Gefährdungen und Maßnahmen.

2.2. Knowledge-Base-Modell

Ein umfangreiches Modell zum Erkennen von potentiellen Innentätern wird von [KMV+10] dargestellt. Die Autoren setzten weniger auf technische Maßnahmen, sondern betrachten persönliche Daten, die mit Hilfe von statistischen Überlegungen und psychologischen Theorien aufgearbeitet werden. Dabei erstellen sie ein detailiertes Bild jedes Angestellten anhand der aktuellen Position („System Role“), seinen Fähigkeiten („Sophistication“), seiner Veranlagung („Predisposition“) und des aktuellen Stress-Levels („Stress Level“).

Zur initialen Einstufung schlagen die Autoren in [KMV+10] vor, durch Befragungen die Werte für Fähigkeiten, Veranlagung und Stress-Level zu bestimmen. Im weiteren Verlauf einer Beschäftigung werden diese Werte durch technische Maßnahmen überprüft und angepasst. Erwähnt wird eine Systemaufrufs-Analyse sowie der Einsatz eines Intrusion Detection Systems und Honeypots. Im vorherigen Abschnitt 2.1 wurden die beiden letzten Maßnahmen schon näher betrachtet. Die Systemaufrufs-Analyse überwacht zum Beispiel, welche Dateien wie oft geöffnet werden. Damit kann im Lauf der Zeit festgestellt werden, ob sich das Verhalten eines Nutzers verändert. Unter Verwendung geeigneter Loggingprogramme würde sich `LoginIDS` auch dazu eignen, anormales Zugriffsverhalten auf Dateien zu erkennen.

Die ermittelten Werte können dann zu verschiedenen Metriken zusammengefasst werden. In [KMV+10] werden Motiv, Gelegenheit und Möglichkeit ermittelt. Daraus wird dann der Wert für die Gesamtbedrohung durch die betrachtete Person berechnet.

Dieses System greift weitreichend in das Arbeitsleben und Privatleben der beobachteten Personen ein. Eine zu detaillierte Beobachtung der Mitarbeiter kann sich aber negativ auf die Moral dieser auswirken. In manchen extremen Fällen zieht das Bekanntwerden solcher Maßnahmen auch einen Ansehensverlust des betroffenen Unternehmens nach sich. Prominente Beispiele, wie die Unternehmen „Lidl“, „Deutsche Telekom“ oder „Die Bahn“, haben durch eine zu enge Überwachung ihrer Mitarbeiter viel negative Aufmerksamkeit in der Presse und Politik erreicht [LÖ9, Kuh10, Haa10].

2.3. Vorschläge des BSI-Grundschutzkatalogs

Das Bundesamt für Sicherheit in der Informationstechnik BSI hat in einem mit etwas mehr als 4000 Seiten starken Katalog zum IT-Grundschutz [Bun11] auch Vorschläge zum Umgang mit Risiken, die durch Mitarbeiter entstehen, gesammelt.

In Abschnitt G3, dem „Gefährdungskatalog Menschliche Fehlhandlungen“, werden viele verschiedene Gefährdungen ausführlich beschrieben. In Abschnitt G3.1 „Vertraulichkeits- oder Integritätsverlust von Daten durch Fehlverhalten“ wird zum Beispiel eine Vielzahl an Möglichkeiten, die zu einem Vertraulichkeitsverlust oder Integritätsverlust führen können, aufgezählt. Darunter fallen Punkte wie: *„Vertrauliche Informationen werden in Hörweiter[sic!] fremder Personen diskutiert, beispielsweise in Pausengesprächen von Besprechungen oder über Mobiltelefonate in öffentlichen Umgebungen.“* [Bun11] oder *„Neue Software wird mit nicht anonymisierten Daten getestet. Nicht befugte Mitarbeiter erhalten somit Einblick in*

geschützte Dateien bzw. vertrauliche Informationen.“ [Bun11] Diese allgemein leicht nachvollziehbaren Beispiele sind gerade im Umfeld einer Universität schwer umzusetzen, da konkrete Beispiele zum anschaulichen Erklären und Zeigen von Sachverhalten wichtig sind.

Andere Punkte umfassen Themen wie G3.17 „Kein ordnungsgemäßer PC-Benutzerwechsel“ oder G3.94 „Fehlkonfiguration der Samba-Kommunikationsprotokolle“. An den hier beispielhaft aufgeführten Themen ist zu erkennen, dass der Katalog sehr konkret potentielle Gefährdungen aufzeigt, die in einem Unternehmen auftreten könnten.

Maßnahmen zur Behandlung werden im Abschnitt M3, dem „Maßnahmenkatalog Personal“, [Bun11] erläutert. Dabei werden allgemeine Vorschläge, wie zum Beispiel im Abschnitt M3.1 zur „Geregelte Einarbeitung/Einweisung neuer Mitarbeiter“, als auch spezielle Maßnahmen, wie die „Schulung zur Lotus Notes Systemarchitektur für Administratoren“ in Abschnitt M3.24 behandelt. Einige der allgemein gefassten Maßnahmen, wie zum Beispiel das Einführen neuer und dem Ausscheiden ehemaliger Mitarbeiter, haben Ähnlichkeit mit Konzepten, die auch in ISO/IEC 27001 A.8 [ISO05] vorgestellt werden.

Der Abschnitt M3.33 „Sicherheitsüberprüfung von Mitarbeitern“ schlägt vor, Referenzen und den Lebenslauf eines Bewerbers gründlich zu prüfen. Im Hinblick auf das in Kapitel 2.2 vorgestellte Knowledge-Base-System und dessen mögliche gesetzliche Grenzen wird eingestanden: „*Die Möglichkeiten, die Vertrauenswürdigkeit von neuem oder fremdem Personal überprüfen zu lassen, sind in Deutschland, aber auch in vielen anderen Ländern, rechtlich sehr eingeschränkt.*“[Bun11] Eine Überwachung während der Anstellung wird in keiner der aufgeführten Maßnahmen behandelt.

Da die im IT-Grundschutz-Katalog beschriebenen Gefahren und Maßnahmen nur Möglichkeiten aufzeigen, aber keine konkreten Verfahren oder Programme vorschlagen, ist er alleine nicht ausreichend. Durch seinen großen Umfang ist er eher dazu geeignet, möglicherweise noch nicht beachtete Sicherheitsrisiken auszumachen und erste Ideen für eine Behandlung der Risiken zu bekommen.

2.4. Fazit

Grundsätzlich kann eine (Paketfilter-)Firewall die Basis einer jeden Sicherheitsarchitektur sein. Damit können auf den Schichten drei und vier des OSI-Schichtenmodells Filterregeln implementiert und ein großer Teil des anfallende Netzwerkverkehr gefiltert werden. Diese Regelsätze bieten aber nur eine grobe Filterung. Außerdem kann die Erstellung von komplexen, möglichst viele Spezialfälle umfassenden Filtertabellen schnell zu Fehlern führen. Intrusion Detection Systeme (IDS) können mit Hilfe von Signaturen oder Heuristiken gezielter Angriffe erkennen und unter Umständen auch direkt blockieren. Diese Erkennung benötigt allerdings mehr Ressourcen als eine reine Firewall. Netzbasierte Intrusion Detection Systeme lesen zum Beispiel über einen Mirror Port den gesamten Netzverkehr mit. Dadurch sind sie anfällig für Überlastungen und müssen eventuell wichtige Pakete verwerfen, die dann nicht analysiert werden können. Hostbasierte Intrusion Detection Systeme hingegen werden auf jedem zu überwachenden Host installiert und können daher sehr genaue Informationen liefern. Durch

die Bindung an einen Host bieten diese Systeme aber selber auch eine gewisse Angriffsfläche. Bei einem Denial of Service (DoS) Angriff fällt mit dem Host-System auch das IDS aus.

Um festzustellen, ob sich Schadprogramme oder zu neugierige Mitarbeiter im Netz bewegen, kann ein Honeypot verwendet werden. Durch die Präsentation eines interessanten Zieles bietet ein Honeypot die Chance einen Angreifer zu enttarnen, der möglicherweise eine Verschlüsselungs- oder Tunneltechnik nutzt, um den zuvor beschriebenen Systemen zu entgehen.

Um mögliche Angriffsvektoren auf das eigene Unternehmen zu erkennen und bewerten zu können, kann ein Katalog wie der IT-Grundschutz-Katalog des BSI verwendet werden. Die darin enthaltene Liste von Gefährdungen kann jedem Unternehmen, das sich um seine IT-Sicherheit sorgt, als Hilfestellung dienen.

Alle diese etablierten Sicherheitsmaßnahmen bieten einen gewissen Schutz vor Angriffen, können aber nicht alle Situationen abdecken. Besonders gefährlich ist das falsche Verhalten von Mitarbeitern, die Zugang zu den zu schützenden Systemen haben. Ein prominentes Beispiel ist die Infektion von iranischen Atomanlagen mit Stuxnet. Anscheinend konnte ein Zutrittsberechtigter über einen infizierten USB-Stick, die Schadsoftware dort installieren [Eik12]. Bei der vorangegangenen Suche nach Schwachstellen in den Systemen, hätte ein heuristisches System unter Umständen Alarm schlagen können.

Um die von den meisten Diensten ständig geschriebenen Logfiles nicht erst reaktiv nach einem Angriff genauer untersuchen zu müssen, sollten diese schon präventiv erkennend auf Anomalien untersucht werden. Im schlimmsten Fall, wenn ein Sicherheitsverstoß erst nach mehrere Wochen oder Monaten auffällt, sind die zur Aufklärung nötigen Logfiles schon aus Datenschutzgründen gelöscht worden. Diese Lücke füllt `LoginIDS`, indem es verdächtige Logeinträge sofort hervorhebt und mit dem Datenschutzrecht vereinbar speichert.

3. Anforderungsanalyse und Konzepte zur Weiterentwicklung

In diesem Kapitel wird zunächst die Entwicklung des `LoginIDS`-Prototypen genauer beschrieben. Auf dieser Basis werden dann Anforderungen an die neue Version aufgestellt. Am Ende des Kapitels wird eine Liste aller neuen Anforderungen erstellt. Diese gibt einen Überblick darüber, welche Anforderungen in dieser Arbeit umgesetzt werden und wo der Entwurf beziehungsweise die Implementierung dieser Anforderungen beschrieben ist.

3.1. Detailinformationen zum `LoginIDS`-Prototypen

Nachdem in Kapitel 1.3 schon allgemein die Funktionalität des `LoginIDS`-Prototypen skizziert wurde, wird vor den Erweiterungen im Rahmen dieser Arbeit hier noch einmal detaillierter auf die Anforderungen und die Funktionalität des Prototypen eingegangen.

3.1.1. Anforderungen

Die Anforderungen und Überlegungen, die zur Entwicklung des `LoginIDS`-Prototypen geführt haben, wurden in dem Bericht [vEMH12b] beschrieben. Zur Vollständigkeit der Beschreibung und der Entwicklung von `LoginIDS` werden hier die wichtigsten Punkte zusammengefasst.

Das Hauptziel war die Entwicklung eines Systems, das einfach in schon bestehende Dienste eingebunden werden kann und unter Berücksichtigung der geltenden Datenschutz- und Mitarbeiterrechte einen Beitrag zur Angriffserkennung leisten kann. Dazu soll das Programm das Nutzerverhalten der beobachteten Benutzer erlernen und Abweichungen von dem „normalen“ Verhalten melden. Dadurch soll die Zahl der von Hand zu untersuchenden Logeinträge entscheidend gesenkt werden. Damit die Erkennung auch für Verbindungen, die von dynamisch zugewiesenen IP-Adressen ausgehen, funktioniert, können konfigurierbare Quelladressbereiche auf einen Namen abgebildet werden. Dazu können sowohl reguläre Ausdrücke als auch DNS zum Einsatz kommen. Detaillierte Informationen zu den beobachteten Verbindungen sollen nach sieben Tagen gelöscht werden. Damit die Erkennungsleistung nicht unter dieser Maßnahme zum Datenschutz leidet, werden die Daten, die zu einer aussagekräftigen Erkennung benötigt werden, aggregiert gespeichert. Dabei ist ein Login zeitlich nur auf eine Stunde genau einzuordnen. Zudem werden IP-Adressen, wenn möglich, durch eine Übersetzung auf einen allgemeineren Namen abgebildet.

3.1.2. Funktionsweise

`LoginIDS` soll durch das Analysieren von Logfiles verschiedener Dienste wirksam die Anzahl der durch einen Menschen zu überprüfenden Logins reduzieren. Dazu wird, unter Berücksichtigung von Datenschutzaspekten, für jeden Nutzer eine Statistik über seine Loginvorgänge erstellt. Anhand dieser Statistik können bei der Analyse verdächtige Logins aus der großen Masse an Logeinträgen heraus gefiltert werden.

Das Programm geht dabei in zwei Schritten vor. Zuerst werden aus den Logfiles der überwachten Dienste die relevanten Felder extrahiert und in ein normalisiertes Format gebracht. Im zweiten Schritt werden diese Daten eingelesen und analysiert. Dazu wird eine Datenbank verwendet, in der die Informationen aus vorangegangenen Logins gespeichert werden.

Diese Datenbank speichert bekannte Benutzer in einer eigenen Tabelle inklusive eines Zeitstempels des ersten und letzten Logins. Ereignisse aus den Logfiles werden in den Tabellen `LongtimeLog` und `DetailedLog` gespeichert. Das `LongtimeLog` dient dabei zur langfristigen und aus Datenschutzgründen an personenbezogenen Daten möglichst reduzierten Speicherung. Prinzipiell dient es dazu, jedem Benutzer die von ihm verwendeten Quell-, Zieladressen-Paare zuzuordnen. Das `DetailedLog` hingegen enthält alle zur Untersuchung eines Vorfalls vorhandenen Daten. Die Zeitpunkte der Zugriffe werden nicht direkt im `LongtimeLog` gespeichert, sondern in einer separaten Tabelle `AccessTimeDetails`. Dadurch können zu einem Quelle-Ziel-Paar beliebig viele Zugriffszeitpunkte gespeichert werden.

Im Rahmen der Anonymisierung im `LongtimeLog` wird eine weitere Tabelle mit dem Namen `NetworkConfiguration` verwendet. Diese Tabelle enthält Regeln, anhand derer bestimmte IP-Adressbereiche unter Verwendung von DNS aufgelöst werden. Im `LongtimeLog` wird dann anstelle der IP nur noch der Name des Adressbereiches verwendet. Die von der Deutschen Telekom an Internetnutzer vergebenen IP-Adressen lassen sich zum Beispiel zu `*.dip.t-dialin.net` auflösen. Durch diese Technik ist es immer noch möglich festzustellen, ob ein Login aus dem selben Netzbereich kommt oder zum Beispiel aus dem Ausland. Diese Technik schützt außerdem vor Fehlalarmen. Da ein regulärer Internet-Heimanschluss alle 24 Stunden vom Internetprovider eine neue dynamische IP-Adresse zugewiesen bekommt, würde ohne die Auflösung jeden Tag eine neue Verbindung erkannt und ein Alarm ausgelöst werden. Das Programm könnte daher kein Normalverhalten lernen. Das `DetailedLog` enthält hingegen noch die echte IP-Adresse, um in einer Untersuchung die vollständigen Informationen zur Verfügung zu haben. Im Rahmen des internen Unternehmensnetzes ist eine solche Funktion aber möglicherweise unerwünscht. Zum Beispiel, wenn über die IP-Adresse einzelne Arbeitsplätze identifiziert werden können und es für einen Mitarbeiter ungewöhnlich ist, einen Dienst von verschiedenen Arbeitsplätzen aus zu nutzen. Dazu kann in der Tabelle `NetworkConfiguration` angegeben werden, welche IP-Adressen übersetzt werden sollen und welche nicht.

Bei der Analyse wird zunächst überprüft, ob ein erkannter Benutzer schon in der Datenbank gespeichert ist. Falls nicht, wird er angelegt und ein `NEW USER` Alarm ausgelöst. Wenn der Benutzer schon bekannt ist, wird das Feld mit dem `lastSeen`-Zeitstempel aktualisiert. Danach wird überprüft, ob das Tupel aus Quell- und Zieladresse für diesen Benutzer schon

existiert. Ist es bekannt, wird auch hier der `lastSeen`-Zeitstempel aktualisiert und ein Zähler für die Anzahl der beobachteten Verbindungen erhöht. Wenn eine unbekannte Kombination von Quell- und Zieladresse erkannt wird, wird ein neuer Eintrag im `LongtimeLog` angelegt und ein `FIRST LOGIN` Alarm gemeldet. Nach dem gleichen Schema wird danach überprüft, ob es zu dem `LongtimeLog`-Eintrag schon einen `AccessTimeDetails`-Eintrag gibt. Falls keiner vorhanden ist, wird ein neuer erstellt und ein Alarm ausgegeben. Ansonsten wird auch hier ein `lastSeen`-Zeitstempel aktualisiert und ein Zähler erhöht. Falls der betrachtete Benutzer in der Lernphase ist, wird kein Alarm (außer dem ersten `NEW USER`) ausgegeben. Die folgende Abbildung 3.1 zeigt diesen Ablauf in einem Aktivitätsdiagramm. Die in manchen Fällen aktualisierten Zeitstempel dienen dazu, veraltete Verbindungen zu erkennen und entfernen zu können. Durch die Zähler ist es möglich, einmalig vorgefundene Verbindungen anders zu bewerten als häufiger vorkommende.

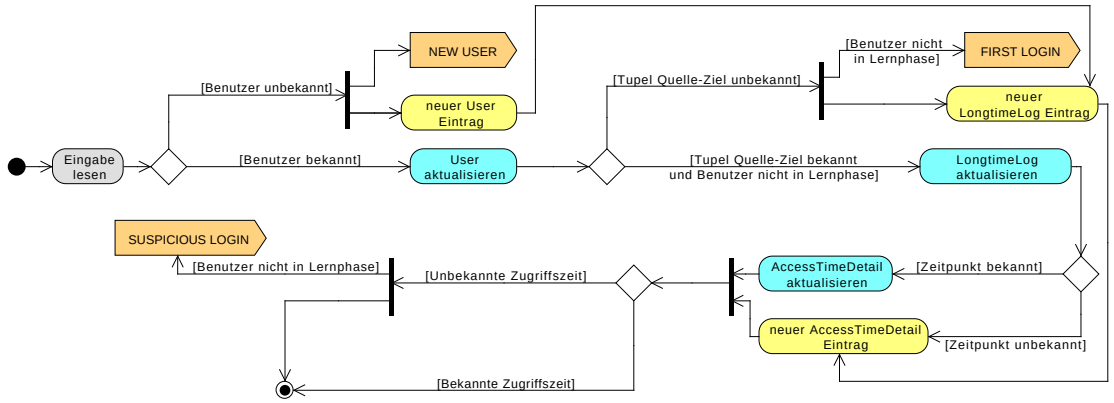

Abbildung 3.1.: Aktivitätsdiagramm für die Analyse des `LoginIDS`-Prototypen

3.1.3. Fazit

Der `LoginIDS` Prototyp erfüllt die an ihn gestellten Anforderungen teilweise. Die Reduktion der Logeinträge funktioniert gut, in ersten Tests mit dem Prototypen wurde eine Reduktion der zu betrachtenden Logeinträge um 91% erreicht [vEMH12a]. Noch nicht umgesetzt ist die geplante Löschung von detaillierten Logeinträgen, die älter als sieben Tage sind. Gerade diese Funktionalität ist aber dringend erforderlich, wenn das Programm in einer realen Umgebung mit echten Kundendaten eingesetzt werden soll.

3.2. Erweiterte Anforderungen an `LoginIDS`

Wie in Abschnitt 1.1 der Einleitung beschrieben, wird bei der Entwicklung dieses Programms speziell auf die Anforderungen des Leibniz-Rechenzentrums eingegangen. In diesem Abschnitt werden die zentralen Anforderungen aus der Aufgabenstellung und Anforderungen, die sich im Laufe des Testbetriebes ergeben haben, vorgestellt.

3.2.1. Unterstützte Logformate

Im Rahmen dieser Arbeit sollen `SSH`-Server-, Windows-Terminalserver-Logs und Apache-Webserver-Logs hinsichtlich der stattfindenden Logins untersucht werden. Für jeden Typ von Logfile wird ein Skript erstellt, das das Parsen des Logfiles übernimmt und die gefundenen Werte in die Datenbank schreibt.

Die Skripte, die aus `SSH`-Server- und Windows-Terminalserver-Logs die relevanten Felder extrahieren, existieren schon für den Prototypen. Um nun zusätzlich Apache-Logs verarbeiten und zukünftig auf einfache Weise neue Dienste überwachen zu können, soll dieser Prozess vereinfacht werden. Zu Beginn war geplant ein einzelnes Programm, das für möglichst alle Logfiles funktionieren sollte, zu entwickeln. Es sollen möglichst viele grundsätzliche Typen von Logfiles unterstützt werden. Das heißt, dass sowohl Logdateien mit einzeiligen als auch mehrzeiligen Logeinträgen unterstützt werden sollen. Über eine Konfigurationsdatei sollten die einzelnen Logdateien und die zugehörigen Felder angegeben werden. Die Konfiguration dieses Programms wäre aber, um alle Spezialfälle abdecken zu können, komplizierter als eines der bestehenden simplen Programme an ein neues Logformat anzupassen. Daher soll ein modularer Ansatz, bei dem für jeden Logfiletyp ein eigenes Konverterprogramm zuständig ist, verfolgt werden.

Der prinzipielle Aufbau eines solchen Programms wird in 4.1 gezeigt. Auf die Schritte, die nötig sind, um aus dem bestehenden `SSH`-Server-Konverter ein passendes Programm für Apache-Logfiles zu schreiben, wird in Kapitel 5.9 eingegangen.

3.2.2. Integration in `Splunk`

Damit die Ausgabe von `LoginIDS` leichter zu analysieren ist, wird eine Integration in das Monitoring- und Analyseprogramm `Splunk` angestrebt. Nähere Informationen zu `Splunk` gibt es im Kapitel 6.1. Dieses Programm kann prinzipiell jede Art von Logfile einlesen und grafisch aufbereitet anzeigen. So ist es zum Beispiel möglich, dynamisch Grafiken über Fehllogins in Abhängigkeit zur Zeit anzuzeigen. Die Möglichkeit innerhalb einer bewährten grafische Oberfläche nach bestimmten Eigenschaften eines Logeintrages filtern zu können, erleichter die Analyse. Zudem wird `Splunk` schon in anderen Bereichen des LRZ eingesetzt, so dass Zusammenhänge zwischen verschiedenen Logfilequellen aufgezeigt werden können.

Innerhalb von `Splunk` muss dann eine geeignete Darstellung der gesammelten Informationen stattfinden. Die Bearbeitung dieser Aufgabe wird in Kapitel 6.2 behandelt.

3.2.3. Verwaltung der erhobenen Informationen

Da LoginIDS auch Informationen sammelt, die nicht nur zur Überprüfung von möglicherweise korrumpierten Accounts geeignet sind, sondern zum Beispiel auch zur Kontrolle der Arbeitszeit eines Mitarbeiters, müssen Überlegungen getroffen werden, wie lange diese Informationen gespeichert werden müssen, beziehungsweise gespeichert werden dürfen. Es muss also in der Datenbank auch eine Art „Vergessen“ vorgesehen sein. Bei der Implementierung muss auch berücksichtigt werden, dass nicht nur aus der LoginIDS Datenbank alte Einträge gelöscht werden, sondern abgelaufene Einträge auch aus dem Index von Splunk entfernt werden.

Unter dem Gesichtspunkt der Sicherheitsanalyse ist es ebenfalls wichtig, alte Daten nicht mehr zu verwenden. Falls zum Beispiel ein Dienst, der vorher häufig, dann aber über einen längeren Zeitraum nicht mehr genutzt wurde, plötzlich wieder verwendet wird, sollte ein Alarm über eine „Erst-Verbindung“ ausgelöst werden. Ein solcher Fall könnte zum Beispiel darauf hindeuten, dass ein Mitarbeiter, der in eine andere Abteilung gewechselt ist oder eine andere Aufgabe zugewiesen bekommen hat, möglicherweise unberechtigt auf Systeme seines alten Arbeitsplatzes zugreift. Mit einer analogen Begründung sollten auch Benutzer nach einer gewissen Zeit aus der Datenbank entfernt werden.

Die „gewisse Zeit“, ab der Einträge aus der Datenbank gelöscht werden, wird auch als Retentionsintervall bezeichnet. Es soll dabei möglich sein, für unterschiedliche Benutzergruppen unterschiedliche Intervalle festzulegen. Eine genaue Betrachtung der gesetzlichen Vorschriften und deren Umsetzung wurde für das LRZ in [MHR11] durchgeführt. Daraus ergibt sich auch die Standardeinstellung, die Detailinformationen nach sieben Tagen löscht. Wie eine solche Retentionsfunktion implementiert wird, ist in Kapitel 4.5 beschrieben.

3.2.4. Unterschiedliche Benutzertypen

In einer Umgebung mit vielen Benutzern ist es von Vorteil, verschiedene Benutzer unterschiedlich zu behandeln. Zum einen könnte für eine Benutzergruppe, die einen Account nur sehr kurze Zeit verwendet, die Lernphase kürzer als normal sein. Bei Benutzern mit geringer Sicherheitsrelevanz könnte eine etwas längere Lernphase verwendet werden, um die Zahl der Alarmmeldungen zu minimieren. Andere Kennungen, wie zum Beispiel „root“, sollen generell einen Alarm auslösen. Am LRZ ist es unüblich, den „root“- oder „administrator“-Account direkt zu verwenden. Personen, die administrativen Zugang benötigen, bekommen zum Beispiel unter Linux über die „sudo“-Funktion höhere Rechte. Daher sollten diese speziellen Systemaccounts von der Verhaltenserkennung und Lernphase ausgenommen und immer gemeldet werden. Das genaue Gegenteil könnte ein Test- oder Administrator-Account sein, der nie einen Alarm auslösen soll und generell ignoriert wird. Für besonders kritische Systeme gibt es möglicherweise Benutzeraccounts, für die es besondere Vereinbarungen gibt, die eine längere Speicherung von Detailinformationen erlaubt.

Welche Benutzertypen im Speziellen sinnvoll wären, wird in Kapitel 4.10 überlegt.

3.2.5. Aggregation der Logfiles von verschiedenen Hosts

Die von LoginIDS zu analysierenden Logfiles müssen von verschiedenen Hosts gesammelt werden. Damit dies auch mit der in Abschnitt 3.2.6 beschriebenen Anforderung eines Real-Time-Monitoring funktioniert, ist es nicht ausreichend, einmal am Tag die Logfiles zu kopieren. Stattdessen sollte es möglich sein, das neue Log-Ereignisse, zeitnah an den LoginIDS-Host übergeben werden.

Eine Möglichkeit dazu wäre es, schon bestehende Software wie das Secure Copy (SCP) zu verwenden, die eine sichere Übertragung der Logdaten über das Netzwerk ermöglicht. Um SCP nutzen zu können, muss jeder Dienst einen SSH-Zugang auf dem LoginIDS-Host bekommen, über den er seine Logdateien abliefern kann. Wie so ein SSH-Zugang abgesichert werden kann, damit keine Logdateien von fremden Diensten gelesen oder Befehle ausgeführt werden können, wird in Kapitel 4.6.1 betrachtet. Ein Vorteil der Lösung mit SCP ist die einfache Konfiguration und die weitere Verbreitung von SSH und SCP, die auf den meisten Linux Distributionen Standardprogramme sind. Aber auch für Windows sind SSH/SCP-Clients verfügbar. Dadurch müsste auf der Seite des Dienstanbieters, der die Logfiles auf den LoginIDS-Host kopieren möchte, unter Umständen keine neue Software installiert werden.

Eine alternative Methode wäre die Verwendung von Programmen wie rsyslog oder syslog-ng. Dadurch würde es vermieden werden, das Netzwerk durch überflüssiges Kopieren großer Logdateien auszulasten und dann extra überprüfen zu müssen, welcher Teil des Logfiles schon verarbeitet wurde und welche Einträge neu sind. Dafür ist der Konfigurationsaufwand höher. Wie eine Umsetzung mit rsyslog aussehen könnte, wird in 4.6.2 beschrieben.

3.2.6. Real-Time Monitoring

Ein großer Nachteil der Verarbeitung, wie sie im Prototypen realisiert war, ist, dass nur alle 24 Stunden eine Analyse läuft. Besser wäre es, die auftretenden Ereignisse in Echtzeit zu behandeln. Wenn die zeitliche Differenz zwischen dem Auftreten und der Analyse eines Ereignisses kleiner wird, könnte die Überprüfung der Alarme zeitlich flexibler stattfinden. Durch die tägliche Analyse um 5 Uhr sind die meisten Ereignisse schon mehr als 12 Stunden alt, bevor sie vollständig analysiert und in Splunk angezeigt werden.

Diese Anforderung hängt sehr stark mit der in Kapitel 3.2.5 beschriebenen Aggregation von Logfiles zusammen. So würde sich für eine Real-Time-Übertragung eine rsyslog- oder syslog-ng-Implementierung auf Grund des geringeren Overheads besser eignen als das zeitgesteuerte Kopieren ganzer Logfiles via SCP. Eine ausführlichere Betrachtung ausgewählter Methoden wird in Kapitel 4.6 beschrieben.

3.2.7. Mehrere Dienste auf demselben Host

Es ist durchaus nicht ungewöhnlich, dass auf einem Host mehrere Dienste parallel laufen. Zum Beispiel könnte ein Apache-Webserver parallel mit einem SSH-Server zur Administration betrieben werden. Damit mehrere Dienste auf einem Host mit LoginIDS analysiert werden

können, muss bei der Zuordnung von Loginvorgängen nicht nur das Tupel aus Quell- und Zieladresse, sondern auch der angesprochene Dienst gespeichert werden. Der Dienst soll dabei über sein Logfile identifiziert werden. Mehr Informationen und die Umsetzung dieser Anforderung werden in Kapitel 4.1 beschrieben.

3.2.8. Zeitfenster

Bei der Bestimmung des Zugriffszeitpunktes wird momentan einfach der Stundenwert des Zeitstempels extrahiert. Dadurch fällt aus Sicht der Analyse durch LoginIDS ein Login um 15:00 Uhr in den gleichen Zeitrahmen wie ein Login um 15:59 Uhr. Analog werden für 14:59 Uhr und 15:00 Uhr unterschiedliche Einträge in AccessTimeDetails angelegt. Das Weglassen der Minuten innerhalb der Datenbank soll zu einem besseren Datenschutz führen. Hier könnte ein Verfahren, dass zusätzlich zur Stunde auch grob die Minuten speichert, die Erkennung verbessern ohne den Datenschutz zu gefährden.

Verschiedene Methoden dazu werden in Kapitel 4.12 anhand von Logdaten untersucht.

3.2.9. Logout festhalten

Neben dem Login sind unter Umständen auch der Logout und die dazugehörige Sitzungsdauer für die Analyse interessant. Wo ein regulärer Benutzer meistens eher einen Großteil des Arbeitstages mit einem Dienst, den er zur Erfüllung seiner Aufgabe benötigt, verbunden ist, könnte eine viel kürzere Verbindung auf einen unbefugten Zugriff zur Sondierung des Systems und der Validierung der Zugangsdaten hinweisen. Im Spezialfall SSH könnte eine kurze Verbindung auch auf einen Kopiervorgang mittels SCP hinweisen. Ebenso könnte eine deutlich längere Sitzung verdächtig sein. In der Zeit könnten langsame Netzwerkscans oder Brute-Force-Angriffe auf andere Accounts laufen. Neben diesen Angriffszenarien gibt es für diese Verhaltensmuster aber auch völlig legitime Gründe. Da die Erkennung und Zuordnung eines Logouts, die in Kapitel 4.1.3 genauer untersucht wird, in den meisten Fällen nicht möglich ist, wird diese Anforderung innerhalb dieser Arbeit nicht umgesetzt.

3.2.10. Anzahl gleichzeitiger Logins

Eine weitere Messgröße, die von LoginIDS überprüft werden könnte, ist die Anzahl der Sitzungen, die pro Benutzer parallel laufen. Ähnlich wie die im vorherigen Abschnitt 3.2.9 vorgeschlagene Überprüfung der Sitzungsdauer, kann auch die Anzahl der Verbindungen wichtige Informationen liefern. Zum Beispiel können bei SSH mehrere Sitzungen gleichzeitig geöffnet sein. Dabei kann aus dem Logfile nicht direkt zwischen Pseudo-Terminals, forwarding oder SCP unterschieden werden. Da diese Anforderung im engen Zusammenhang mit der Logout-Erkennung aus Abschnitt 3.2.9 steht, ist diese Anforderung erst mal nur eine theoretische Überlegung, die hier nicht umgesetzt wird.

3.2.11. Erweiterte Lernphase

Die Lernphase ist im Prototypen auf einen Monat festgelegt. Bei dem Beginn des `LoginIDS` Testbetriebs im Dezember 2011 trat das Problem auf, dass viele Nutzer während der Lernphase in den Weihnachtsferien waren. Nach Ablauf der Lernphase wurden dadurch viele Warnmeldungen erzeugt, da `LoginIDS` das Verhalten vorher nicht ausführlich genug analysieren konnte. Abhilfe könnte eine Anpassung der Kriterien für die Lernphase bieten.

In Kapitel 4.8 wird beschrieben, wie die Konfiguration der Lernphase funktioniert. In Kapitel 4.13 wird untersucht, welche Möglichkeiten es gäbe, die Lernphase individueller zu gestalten.

3.2.12. IPv6 Unterstützung

Nachdem die Unterstützung von IPv6, dem Nachfolge-IP-Protokoll von IPv4, in vielen Netzen und mit vielen Diensten selbstverständlich wird, muss `LoginIDS` auch damit umgehen können. Im Münchner Wissenschaftsnetz (MWN) wird IPv6 auch schon aktiv eingesetzt und auch immer häufiger genutzt [Lei11, Abschnitt 3.2.12]. Der einzige nicht IPv6 kompatibel Teil ist dabei die DNS-Auflösung. An anderen Stellen werden IP-Adressen als Strings behandelt, um diese im selben Datenbankfeld wie Domainnamen speichern zu können. Wie die DNS-Auflösung verändert werden kann, um sowohl für IPv4 als auch IPv6 zu funktionieren, wird in Kapitel 5.5.2 gezeigt.

3.2.13. Arbeitsplatzerkennung

Innerhalb des Arbeitsplatznetzes am LRZ werden den Workstations über DHCP IP-Adressen zugeteilt. Diese Zuteilung ist für ungefähr eine Woche gültig. Damit zuverlässig erkannt wird, ob ein Benutzer einen Dienst von seinem Arbeitsplatz oder von dem eines Kollegen aus benutzt, haben die Arbeitsplätze eindeutige Namen, die über DNS abgefragt werden können. Die Konfiguration für welche Netze eine DNS-Auflösung stattfinden soll, wird in der `NetworkConfiguration` Tabelle der `LoginIDS`-Datenbank vorgenommen. Diese Konfiguration wird in Kapitel 4.4.2 definiert.

3.2.14. `MySQL` Datenbank

Damit es einfacher ist Tests durchzuführen und wegen des geringen Konfigurationsaufwandes, wurde bei der Entwicklung des `LoginIDS`-Prototypen eine `SQLite` Datenbank verwendet. Mit steigender Anzahl an Datenbankeinträgen und der Notwendigkeit, mit mehreren Programmen gleichzeitig und effektiv auf der Datenbank arbeiten zu können, wird ein leistungsfähigeres Datenbanksystem benötigt. Dazu wird die Unterstützung des weit verbreiteten `MySQL` zu `LoginIDS` hinzugefügt. Dadurch, dass beide Systeme einen eigenen SQL-Dialekt sprechen, können die meisten Anfragen unverändert übernommen werden. Das Skript zum Erstellen der nötigen Tabellen muss allerdings stark verändert werden, da die Datentypen in `SQLite`

und MySQL unterschiedlich benannt sind. Nach dem Erstellen der Tabellen übernimmt das Perl-Modul DBI, die Umsetzung von Perlvariablen in die jeweiligen Datenbank-Datentypen. Das zum Einrichten der LoginIDS-Konfiguration geschriebene Setup-Skript wird in Kapitel 4.7 detailliert vorgestellt.

3.2.15. Integritätssicherung

Damit das LoginIDS-System selber eine möglichst geringe Angriffsfläche bietet und Manipulationen verhindert werden können oder zumindest auffallen, muss die Integrität des Systems sichergestellt werden. Einfache Maßnahmen sind, dass LoginIDS als eigener Benutzer ausgeführt wird und andere Nutzer auf dem System keinen Zugriff auf die Programmdateien und die Datenbank haben. Für MySQL Datenbank und Splunk, das zur Anzeige der LoginIDS Analyse verwendet wird, existieren ebenfalls Authentifikationsmechanismen, die vor ungewollten Zugriffen schützen. Die Programmdateien könnten zusätzlich durch kryptographische Prüfsummen gesichert werden, die vor jeder Ausführung automatisch überprüft werden. Für eine erste Version von LoginIDS, die auf einer dedizierten virtuellen Maschine ausgeführt wird, ist eine solche Überprüfung nicht vorgesehen. In zukünftigen Versionen könnte ein solcher Mechanismus umgesetzt werden.

3.2.16. Usermapping

Da eine Person unter Umständen mehrere Benutzerkennungen hat, wäre es praktisch diese geeignet auf einander abzubilden. Zum Beispiel können so zwei auf den ersten Blick unabhängige Warnmeldungen der selben Person zugeordnet werden. Bei der weit verbreiteten Praxis, ein Passwort für mehrere verschiedene Dienste zu verwenden, wäre bei einem kompromittierten Passwort zu erwarten, dass mehrere Benutzerkonten einer Person, für die das gleiche Passwort verwendet wurde, zur gleichen Zeit Warnmeldungen generieren.

Da eine solche Abbildung nicht automatisiert werden kann und es das Ziel von LoginIDS ist, den Administrationsaufwand bei der Logfile-Analyse zu senken, wird ein solches Mapping im Rahmen dieser Arbeit nicht umgesetzt.

3.2.17. Validierung von Benutzernamen

Die Benutzernamen im LRZ folgen im Allgemeinem einem bestimmten Schema, das durch einen regulären Ausdruck auf seine Gültigkeit hin untersucht werden könnte. Darüber könnten Tippfehler in einem Benutzernamen automatisch erkannt werden. Der Aufbau des Benutzernamens ist wie folgt: Zwei Buchstaben gefolgt von zwei Ziffern sowie drei weiteren Buchstaben und einer optionalen Ziffer am Ende. Der reguläre Ausdruck dafür könnte so aussehen: `\w{2}\d{2}\w{3}\d?`. In dem Schema sind nicht alle Buchstaben erlaubt, um ähnlich aussehende Zeichen zu vermeiden. Zusätzlich werden als Buchstabenkombinationen, zur besseren Diktierbarkeit, nur Silben verwendet. Auch dafür ließe sich ein passender aber weitaus komplizierterer regulärer Ausdruck schreiben.

Weitere Überlegungen und Lösungen werden in Kapitel 4.11 erläutert.

3.2.18. Ausführen externer Skripte

Damit individueller auf die verschiedenen Alarme reagiert werden kann, soll bei jeder Alarmmeldung ein externes Programm ausgeführt werden können. Diese Skripte werden im Folgenden auch als Trigger-Skripte bezeichnet. Dort können dann spezielle Logfiles mit allen Alarmmeldungen geschrieben werden oder bei bestimmten Alarmen E-Mails versandt werden. Für den Alarm `NEW USER` kann das externe Skript auch eine Rückgabe an `LoginIDS` geben und dadurch eine externe Benutzervalidierung ermöglichen, die im vorhergehenden Abschnitt 3.2.17 gefordert wurde. Dieser Rückgabewert definiert dann den Benutzertyp. Der Benutzertyp kann über eine Benutzerverwaltung, für die am LRZ ein LDAP-Verzeichnis verwendet wird, oder eine beliebige andere Methode bestimmt werden. In Kapitel 4.9 wird der weitere Entwurf dieser Trigger-Skripte gezeigt.

3.3. Liste der Anforderungen

Hier werden die Anforderungen, die im vorherigen Kapitel 3.2 beschrieben wurden, noch einmal in tabellarischer Form dargestellt. Zudem wird angegeben, in welchem Kapitel der Entwurf und die Implementierung beschrieben wird.

Anforderung	Kurzbezeichnung	Entwurf	Implementierung
3.2.1	Logfiles	4.1	5.9
3.2.2	`Splunk`	6.2	6.2
3.2.3	Logfileverwaltung	4.5	5.8
3.2.4	Benutzertypen	4.10, 4.3	5.6, 5.4
3.2.5	Logfileaggregation	4.6	5.2
3.2.6	Real-Time-Analyse	4.6	5.2
3.2.7	Mehrere Dienste auf einem Host	4.1	5.4
3.2.8	Angepasste Zeitfenster	4.12	nicht weiter verfolgt
3.2.9	Logouterkennung	4.1.3	nicht weiter verfolgt
3.2.10	Gleichzeitige Logins	nicht weiter verfolgt	
3.2.11	Erweiterte Lernphase	4.13, 4.8	5.7
3.2.12	IPv6 Unterstüzung	4.4.2	5.5.2
3.2.13	Arbeitsplatzerkennung	4.4.2	5.5
3.2.14	`MySQL` Unterstützung	4.2	5.4
3.2.15	Integritätssicherung	nicht weiter verfolgt	
3.2.16	Benutzermapping	nicht weiter verfolgt	
3.2.17	Benutzervalidierung	4.11	5.10.1
3.2.18	Externe Skripte	4.9	5.10

4. Entwurf

Dieses Kapitel beschäftigt sich mit dem Entwurf der Implementierung der in Kapitel 3 beschriebenen Anforderungen. Es werden mögliche Umsetzungen, deren Aufwand und Machbarkeit diskutiert. Dann wird entschieden, ob die Anforderung in dieser Arbeit umgesetzt wird.

4.1. Einlesen von Logfiles

Die Eingabe für `LoginIDS` sind Logfiles. Um möglichst universell einsetzbar zu sein, müssen möglichst viele verschiedene Logfiletypen analysiert werden können. Dazu wird das System, das die Konvertierung übernimmt, modular aufgebaut. Dies und die Alternative eines komplexen Systems, das möglichst alle Logfiles erfassen kann, wurde in Kapitel 3.2.1 zur Anforderungsanalyse diskutiert.

Das modulare System arbeitet mit einem eigenen Konvertierungsskript pro Logfileformat. Zu diesem System gehört eine besondere Ordnerstruktur, bei der die Skripte im Ordner `converter-available` liegen und bei Bedarf per Symlink in den Ordner `converter-enabled` verlinkt werden. Über den Namen des Links in dem `converter-enabled` Ordner wird auch gesteuert, für welche Logfiles dieser Konverter verwendet wird. Das Namensformat der Logfiles muss dabei `<Typ des Logfiles>-<Hostname>-<Timestamp>.<Endung>` entsprechen und der Link zu dem zu verwendenden Konverter muss dann entsprechend `<Typ des Logfiles>` heißen. Die Lösung mit einem Link erleichtert es auch, zwei Logfiletypen mit dem selben Konverter abarbeiten zu können. Dazu müssen nur zwei Links mit unterschiedlichen Namen angelegt werden, die auf den selben Konverter verweisen. Der `<Timestamp>` kann dabei nur eines der folgenden Formate haben: „`%Y-%m-%d`“ oder „`%Y-%m-%dT%H:%M:%S`“. Die Formatierungszeichen entsprechen den von dem Linux-Befehl `date` bekannten Format. Konkret sehen die Zeitstempel also so aus: „`2012-01-25`“ oder „`2012-01-25T14:37:13`“.

Die Konverterprogramme werden von dem `LoginIDS`-Daemon `logindexd` aufgerufen. Dieser Daemon läuft ständig im Hintergrund und erkennt, wenn neue Logdateien im `input`-Ordner abgelegt werden. Er sortiert alle verfügbaren Logdateien und ruft die Konverter dann mit den Logdateien in der richtigen Reihenfolge auf.

4.1.1. Sortieren der Logdateien

Zum Einlesen der Logfiles in der zeitlich richtigen Reihenfolge sind folgende Maßnahmen erforderlich. Wenn es mehrere Logfiles des gleichen Dienstes und Hosts gibt, sollen die Ein-

gabedateien in der zeitlich richtigen Reihenfolge an `LoginIDS` übergeben werden. Zum Beispiel sollte es nicht passieren, dass ein Benutzer als neuer Benutzer gemeldet wird, dann aber in einem früheren Logfile noch einmal gefunden wird. Danach würden Loginvorgänge vor dem bisherigen ersten Login aufgezeichnet werden. Dies könnte passieren, wenn die Logfiles aus Abbildung 4.1 von oben nach unten abgearbeitet werden. Die markierten Bereiche bedeuten dabei, dass in dem Logfile zu dieser Zeit Logeinträge gespeichert sind. Damit es keine Anomalien gibt, müsste zuerst Logfile 2, dann Logfile 1 und zum Schluss Logfile 3 gelesen werden. Dazu wird ein festes Format für Eingabedateien angegeben, welches im vorhergehenden Abschnitt 4.1 beschrieben wurde. Anhand dieses Schemas kann über den Zeitstempel die Reihenfolge des Einlesens festgelegt werden. Alternative Methoden wie das Lesen der Modified- oder Created- Zeitpunkte, die ein Dateisystem eventuell bietet, sind an dieser Stelle eher ungünstig. Die Dateien werden von verschiedenen Systemen gesammelt und zusammen kopiert, wobei diese Werte meistens verändert werden. So können auch Dateien, die aus einem Backup wiederhergestellt werden, ohne Zeitanomalien zu verursachen, eingelesen werden. Das Sortieren vor dem Auswerten wird von dem Programm `logindexd` übernommen.

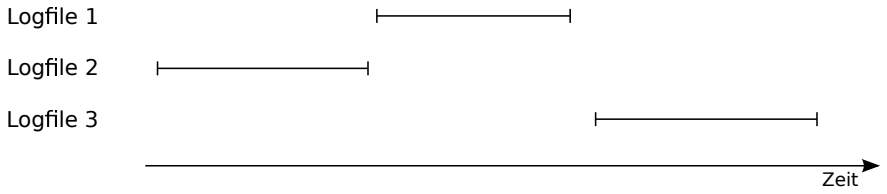

Abbildung 4.1.: Schema zur Reihenfolge beim Einlesen von Logfiles

Damit nicht nur die einzelnen Eingabedateien richtig sortiert werden, sondern auch alle Logfileeinträge, die aus verschiedenen Logfiles stammen können, wird beim Extrahieren der einzelnen Einträge kein neues Zwischen-Logfile geschrieben, sondern die Einträge in eine eigene Datenbanktabelle geschrieben. Dadurch wird auch das Lesen der Eingabe-Logfiles von der Analyse der gelesenen Einträge entkoppelt. Das Analysetool `loginidsd` von `LoginIDS` kann die Logeinträge dann, von der Datenbank richtig sortiert, verarbeiten. Zudem müssen die Programmteile nicht mehr per Cronjob gesteuert gestartet werden, sondern können als Daemon laufen. Dadurch wird vor allem die in Abschnitt 3.2.6 geforderte Real-Time-Analyse einfacher. Dieser prinzipielle Ablauf ist in Abbildung 4.2 dargestellt. Diese Abbildung zeigt auch den Unterschied zu dem vorherigen Ablauf im Prototypen, wie er in Abbildung 1.2 dargestellt wurde.

Trotz der hier beschriebenen Maßnahmen, kann es zu Anomalien kommen. Dies wird im Folgenden anhand eines Beispieles veranschaulicht: Es gibt dabei zwei Dienste, einen `SSH`-Server, der seine Logfiles einmal täglich um 1 Uhr an `LoginIDS` schickt, und einen Citrix-Windows-Terminalserver, der seine Logfiles stündlich analysieren lässt. Ein neuer Benutzer, der unter demselben Login-Namen auf beiden Diensten zugangsberechtigt ist, loggt sich nun zu Beginn der Arbeit um 8 Uhr auf dem `SSH`-Server ein. Erst am Nachmittag kurz nach 15 Uhr verbindet er sich das erste Mal mit dem Citrix-Server. Da der Citrix-Server seine Logfiles stündlich an den `LoginIDS`-Host schickt, wird dort um 16 Uhr ein neuer Benutzer

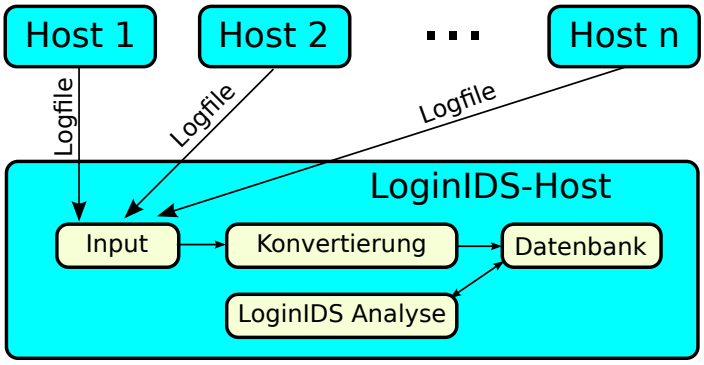

Abbildung 4.2.: Schema für den Ablauf der `LoginIDS` Analyse

erkannt. Erst um 1 Uhr des nächsten Tages, wird das Logfile des `SSH`-Servers analysiert; dort wird der Login des Benutzers von 8 Uhr gefunden, der jetzt vor der Uhrzeit liegt, an der der Benutzer das erste Mal in einem Logfile aufgetreten und in der `LoginIDS`-Datenbank erstellt worden ist. In den meisten Fällen hat diese Anomalie keine weitere Bedeutung. Der `firstSeen`-Zeitstempel, der für einen Benutzer in der Datenbank geführt wird, muss in diesem Fall einige Stunden nach vorne geschoben werden. In extremeren Fällen, wo zum Beispiel nachträglich Logfiles eines ganzen Monats eingespielt werden, kann das dazu führen, dass Benutzer, die gerade von `LoginIDS` als neue Benutzer erkannt wurden und daher gerade in der Lernphase sind, diese sofort wieder verlassen. Diese dargestellten Anomalien sind allerdings keine wirklichen Probleme, die verhindert werden müssten.

Eine wichtige Schlussfolgerung daraus ist, dass die Alarmmeldungen nicht in der Reihenfolge ihres Zeitstempels überprüft werden dürfen. Wenn nachträglich Dateien mit Logmeldungen von früheren Ereignissen gelesen werden, haben die Alarmmeldungen einen früheren Zeitstempel und würden übersehen werden. Daher sollten Alarmmeldungen immer nach ihrer ID sortiert werden. Dort ist sichergestellt, dass wenn alle Alarmmeldungen bis zur ID x überprüft worden sind, keine Alarmmeldungen mit einer ID $< x$ später noch hinzukommen.

4.1.2. Konvertierungsskripte

Ein Konvertierungsprogramm muss die folgenden Informationen aus einem Logfile lesen: einen Zeitstempel, an dem das Ereignis registriert wurde, den Benutzernamen, der das Ereignis ausgelöst hat, Quell- und Zieladresse und ob der Login erfolgreich war oder nicht. Zusätzlich muss jeder Logeintrag noch einem Dienst zugeordnet werden. In Kapitel 4.3 zum Datenbankschema wird diese Tabelle beschrieben.

Die grundlegenden Überlegungen, was ein solches Konvertierungs-Tool können muss, werden zunächst beispielhaft anhand des Logs des `SSH`-Servers „`sshd`“ und später auch anhand des Apache-Webserver-Logs betrachtet. Wenn im Folgenden von Apache gesprochen wird, ist in der Regel auch der Apache-Webserver gemeint.

Die Logeinträge von `SSH` werden häufig mit den Logeinträgen anderer Dienste in Logdateien

wie `/var/log/messages` oder `/var/log/auth.log` gespeichert. Das Listing 4.1 zeigt einen Auszug aus einem `auth.log` von einer Ubuntu-Maschine. Damit die Darstellung übersichtlicher wird, wurde der Zeitstempel und der Hostname, die normalerweise am Anfang jeder Logzeile stehen, entfernt. Zwischen den Meldungen der Programme `cron` und `sudo` finden sich auch Meldungen von `sshd`. Beim Einlesen des Logfiles zum Analysieren der `SSH`-Logins muss daher zuerst festgestellt werden, ob die gelesene Zeile überhaupt zum `SSH`-Dienst gehört. Ein sicheres Indiz, dass ein Logeintrag zu `sshd` gehört, ist es in diesem Fall, wenn im Anfang der Lognachricht „`sshd[xyz]:`“ steht. Das bedeutet, dass der Logeintrag von einem Prozess mit dem Namen `sshd` und der Prozessnummer `xyz` stammt.

Listing 4.1: Auszug aus einem Ubuntu `auth.log`

```
1 CRON[27023]: pam_unix(cron:session): session closed for user root
2 CRON[27021]: pam_unix(cron:session): session closed for user www-data
3 sshd[27589]: Accepted password for jupiter from 192.168.178.41 port 44347 ssh2
4 sshd[27589]: pam_unix(sshd:session): session opened for user jupiter by (uid=0)
5 sudo:  jupiter : TTY=pts/2 ; PWD=/home/jupiter ; USER=root ; COMMAND=/usr/bin/tail
      -n 50 /var/log/auth.log
6 sudo: pam_unix(sudo:session): session opened for user root by jupiter(uid=1000)
7 sudo: pam_unix(sudo:session): session closed for user root
8 sshd[27752]: Received disconnect from 192.168.178.41: 11: disconnected by user
9 sshd[27589]: pam_unix(sshd:session): session closed for user jupiter
```

Das Listing 4.1 zeigt auch, dass nicht jede Zeile, die der `SSH`-Server loggt, auch für die Analyse notwendig ist. Zum Beispiel reicht es zum Erkennen eines erfolgreichen Logins aus, die dritte Zeile einzulesen. Diese zeigt an, dass der Zugriff erfolgreich war, wie der Benutzer heißt und von welcher IP-Adresse er zugegriffen hat. In dem Beispielauszug heißt der Benutzer `jupiter`. Da die Logmeldungen immer einem festen Schema folgen, lassen sich die entsprechenden variablen Felder gut mit regulären Ausdrücken selektieren und abspeichern.

Bei den Logdateien des Apache-Webservers entsteht das Problem, dass bei der Verwendung von `mod_auth` nicht alle Meldungen zur Authentifizierung über `.htaccess` Dateien in einem, sondern zwei Logfiles gespeichert werden. Fehlermeldungen über nicht vorhandene Nutzer oder falsche Passwörter werden standardmäßig in `/var/log/apache2/error.log` gespeichert. Ein Beispiel dazu ist in Listing 4.2 zu sehen. Erfolgreiche Logins werden gar nicht extra aufgeführt, sondern es wird im Zugriffslog zusätzlich ein Benutzername angegeben. Listing 4.3 zeigt hierfür ein Beispiel. Durch diese Aufteilung der Logeinträge scheint die Analyse zunächst komplizierter zu werden. Da es aber prinzipiell egal ist, welche der beiden Dateien wir zuerst lesen und die Einlese-Skripte modular sind, kann ein Skript für das `error.log` und eines für das `access.log` geschrieben werden. Die Implementierung dieser beiden Skripte wird in Kapitel 5.9 beschrieben.

Listing 4.2: Apache-Webserver `error.log`: Beispiele für fehlerhafte Logins

```
1 [Mon Apr 09 11:48:56 2012] [error] [client 192.168.178.41] user foo not found:
      /geschuetzt
2 [Mon Apr 09 11:57:22 2012] [error] [client 192.168.178.41] user michael:
      authentication failure for "/geschuetzt": Password Mismatch
```

Listing 4.3: Apache-Webserver `access.log`: Beispiel für einen erfolgreichen Login

```
1 192.168.178.42:443 192.168.178.41 - michael [09/Apr/2012:12:02:28 +0200] "GET
    /geschuetzt/index.html HTTP/1.1" 200 1213
    "https://192.168.178.42/geschuetzt/index.html" "Opera/9.80 (X11; Linux x86_64;
    U; en) Presto/2.10.229 Version/11.62"
```

4.1.3. Logout Erkennung

In der Anforderungsanalyse wurde in Kapitel 3.2.9 vorgeschlagen, den Logoutzeitpunkt ebenfalls in `LoginIDS` aufzuzeichnen und die Analyse somit um eine Sitzungsdauer zu verfeinern. Diese Idee erweist sich in der Praxis als schwer umzusetzen. In diesem Kapitel wird beschrieben, wo bei `SSH`-Server- und Apache-Webserver-Logs die Probleme auftreten.

Bei dem Apache-Webserver-Log ist das Problem, dass es keinen expliziten Logout gibt. Das Ende einer Sitzung muss also anhand des Zeitstempels des letzen Zugriffes und einem Timeout ermittelt werden. Etwas anders stellt sich die Situation bei dem Log des `SSH`-Servers dar. Der `SSH`-Servers `sshd` schreibt Zugriffslogs in ein Logfile wie `/var/log/messages` oder `/var/log/auth.log`. Hier werden sowohl Login- als auch Logout-Zeitpunkte sowie Fehler gespeichert. Das Problem ist hierbei, zuverlässig den Logout zu erkennen. Über `SSH` können beliebig viele einzelne Verbindungen aufgebaut werden. Abhängig von der `SSH`-Version und dem angegebenen LogLevel wird im Logfile aber ein Logout keinem bestimmten Login, sondern nur einer IP-Adresse zugeordnet. Gelöst werden könnte dieses Problem über ein Mitzählen von Login- und Logoutvorgängen. Es besteht aber nicht die Möglichkeit zu erkennen, welchem Login dieser Logout entgegensteht. Diese Analyse hat daher keinerlei Aussagekraft.

Bei neueren Versionen von `SSH` kann eine `SSH`-Sitzung über die im Logfile aufgeführte Prozess-ID (PID) erkannt werden. In den Listings 4.4, 4.5 und 4.6 werden die Logeinträge verschiedener `SSH`-Versionen und unterschiedlich ausführlicher Logeinstellungen gezeigt. Listings 4.4 und 4.5 zeigen dabei die `SSH`-Version, die momentan in Debian Squeeze(stable) aktuell sind. Listing 4.6 zeigt ein `SSH` aus einem aktuellen Ubuntu 12.04. Ein Logeintrag von `sshd` setzt sich aus „`<Zeitstempel> <Host> sshd[<PID>]: <Meldung>`“ zusammen.

Wie in Listing 4.4 zu sehen ist, kann unter Open`SSH` 5.5p1 in der Standardeinstellung mit LogLevel INFO aus dem Verbindungsabbruch nicht geschlossen werden, dass die Sitzung, die kurz davor gestartet wurde, beendet wird. Es könnte sich auch um eine noch davor vom gleichen Host gestartete Verbindung handeln.

Listing 4.4: OpenSSH_5.5p1 Debian-6+squeeze2, OpenSSL 0.9.8o 01 Jun 2010; LogLevel INFO

```
1 May 21 19:33:52 saturn sshd[16737]: Accepted publickey for michael from
    192.168.178.41 port 33438 ssh2
2 May 21 19:34:13 saturn sshd[16739]: Received disconnect from 192.168.178.41: 11:
    disconnected by user
```

Listing 4.5 zeigt die selbe `SSH`-Version mit einer ausführlicheren Logeinstellung. Hier könnte eine Verbindung zwischen Login und Logout hergestellt werden. In der vierten Zeile wird

ausgegeben, dass ein gültiger Login stattgefunden hat. In der Zeile darauf wird angegeben, welche PID der Prozess hat, der dieser Verbindung zugeordnet ist. In diesem Beispiel 14834. Der Logout in der letzten Zeile kann dann über die PID 14834 dem vorhergegangenen Login zugeordnet werden.

Die einfachste Zuordnung bietet OpenSSH 5.9p1, wie in Listing 4.6 zu sehen ist. Anfang und Ende einer Sitzung sind hier über die zweite und vierte Logzeile direkt zu ermitteln. Die Zuordnung kann hier auch über die PID, im Beispiel 4619, erfolgen.

Listing 4.5: OpenSSH_5.5p1 Debian-6+squeeze2, OpenSSL 0.9.8o 01 Jun 2010; LogLevel VERBOSE

```
1 May 21 19:14:03 saturn sshd[14832]: Connection from 192.168.178.41 port 33025
2 May 21 19:14:03 saturn sshd[14832]: Found matching RSA key:
      23:49:6e:53:70:61:63:65:54:68:61:79:72:61:3a:29
3 May 21 19:14:11 saturn sshd[14832]: Found matching RSA key:
      23:49:6e:53:70:61:63:65:54:68:61:79:72:61:3a:29
4 May 21 19:14:11 saturn sshd[14832]: Accepted publickey for michael from
      192.168.178.41 port 33025 ssh2
5 May 21 19:14:11 saturn sshd[14832]: User child is on pid 14834
6 May 21 19:14:30 saturn sshd[14834]: Received disconnect from 192.168.178.41: 11:
      disconnected by user
```

Listing 4.6: OpenSSH_5.9p1 Debian-5ubuntu1, OpenSSL 1.0.1 14 Mar 2012; LogLevel INFO

```
1 May 20 12:27:34 jupiter sshd[4619]: Accepted password for michael from
      192.168.178.41 port 54864 ssh2
2 May 20 12:27:34 jupiter sshd[4619]: pam_unix(sshd:session): session opened for user
      michael by (uid=0)
3 May 20 12:28:06 jupiter sshd[4715]: Received disconnect from 192.168.178.41: 11:
      disconnected by user
4 May 20 12:28:06 jupiter sshd[4619]: pam_unix(sshd:session): session closed for user
      michael
```

Die am LRZ aktuell verwendete Linux Version ist ein SUSE Linux Enterprise Server 11.1, welcher auf der Testmaschine mit einem `OpenSSH_5.1p1, OpenSSL 0.9.8j-fips 07 Jan 2009` verwendet wird. Dort werden nur Loginvorgänge nach `/var/log/messages` gelogged, Logouts werden in keiner Weise aufgeführt.

Neben der Problematik, dass die echte Dauer einer Verbindung abhängig von der Version schwer oder gar nicht erkennbar ist, könnten auch unsauber geschlossene Verbindungen, die erst nach längerer Zeit durch einen Timeout geschlossen werden, das Ergebnis erheblich verfälschen. Daher werden im Rahmen dieser Arbeit nur Loginzeitpunkte beachtet. Im Datenbankschema, welches in Abschnitt 4.3 detailliert vorgestellt wird, ist ein Feld zum Unterscheiden zwischen Login und Logout aber schon vorgesehen.

4.2. Datenbanksystem

Bei der Wahl der zu unterstützenden Datenbanksysteme wurden `SQLite` und `MySQL` ausgewählt. `SQLite` ist gerade in einer Testumgebung praktisch, da es kaum Konfiguration benötigt und schnell einsetzbar ist. In Hinblick auf die Performance ist `SQLite` aber keine

gute Wahl, da für Transaktionen die gesamte Datenbank gesperrt wird und es keine Locks auf Tabellen- oder Tupel-Ebene gibt. Zusätzlich gibt es kein Benutzermanagement. Jeder, der Zugriff auf die Datenbankdatei hat, kann diese verwenden. In einer Datenbank mit Usermanagement könnte es zum Beispiel einen Benutzer geben, der die Einträge anschauen und Alarme überprüfen kann ohne sie verändern zu dürfen. Ebenfalls nicht möglich ist eine verteilte Struktur, bei der die Datenbank über das Netz angebunden wird.

Für den Fall, dass die oben genannten Einschränkungen von `SQLite` umgangen werden sollen und ein erhöhter Konfigurations- und Management-Aufwand akzeptiert wird, kann auch `MySQL` als Datenbanksystem verwendet werden.

4.3. Datenbankschema

Die Datenbank soll als zentraler Speicher des Programmes dienen, sowie die Verwaltung zur Analyse anstehender Logeinträge erleichtern. Die aus den analysierten Logfiles gewonnen Informationen, werden in der Datenbank hinterlegt. Wichtige Elemente sind dabei Benutzer, Loginvorgänge und ausgelöste Alarme.

Da das System für jeden Benutzer eine individuelle Statistik über Anmeldungen erstellt, soll ein Benutzer und die ihm zugeordneten Ereignisse eindeutig identifiziert werden können. Dazu wird eine eindeutige ID vergeben. Zudem kann jeder User über einen Loginnamen, wie er in den durchsuchten Logfiles vorkommt, erkannt werden.

Um „Datenbankleichen“ erkennen zu können, soll für jeden Benutzer gespeichert werden, wann er zuletzt eingeloggt war. Diese Information kann zudem dabei helfen, zu überprüfen, ob ein Benutzer möglicherweise noch Zugangsrechte hat, die er nicht mehr benötigt. Auch wann ein Benutzer das erste Mal in einem Logfile erkannt und in die Datenbank eingefügt wurde, wird gespeichert. Dieser Zeitstempel ist vor allem wichtig, um festzustellen wie lange die Lernphase dauert. Sollte der Zugriff außerdem unautorisiert gewesen sein, kann der Vorfall durch den Zeitstempel besser analysiert werden. Um für verschiedene Nutzer unterschiedlich strenge Richtlinien vergeben zu können, wird jedem Nutzer zudem ein Benutzertyp zugeordnet. Dieser Typ kann zum Beispiel das Retentionsintervall für Logeinträge oder Schwellwerte für abweichende Loginzeitpunkte beeinflussen. Die Anforderung dazu wurde in Abschnitt 3.2.4 beschrieben.

Neben den Benutzern sollen auch die wichtigen Informationen aus den analysierten Logfiles gespeichert werden. Diese Zusatzinformationen identifizieren einen Service sowie Quell- und Zieladresse der aufgebauten Verbindung. Das Hinzunehmen des Services zu Quell- und Zieladresse ist eine Anforderung, die in Abschnitt 3.2.7 beschrieben ist. Ein zusätzlicher Zähler zeigt an, wie oft dieses Verbindungspaar von diesem Benutzer schon aufgebaut wurde. Diese Information könnte nützlich sein, wenn eine Verbindung erst ab einem Schwellwert als „gut“ bewertet werden soll. Für jedes dieser Verbindungspaare wird auch noch ein Zeitstempel der letzten Verbindung gespeichert. Dadurch können längere Zeit ungenutzte Verbindungen entfernt werden.

Nur den Zeitpunkt der letzten Verbindung zu speichern, reicht aber nicht aus, wenn das

Programm erkennen soll, zu welcher Zeit eine Verbindung wahrscheinlich legitim ist. Daher wird in einer weiteren Tabelle zu jedem Zugriff der ungefähre Zeitpunkt gespeichert. Auch hier existiert wieder ein Zähler, um zu überprüfen, wie oft der Zugriff zu diesem Zeitpunkt erfolgt. In dem Entwurf zu Zeitfenstern in Kapitel 4.12 wird überlegt, wie der Zeitpunkt am geschicktesten zu ermitteln ist. Aufgrund der nahezu Gleichverteilung der Logins werden einfach die Minuten weggelassen und nur der Stundenwert gespeichert.

Sollte ein Logeintrag einen Alarm auslösen, wird dieser auch in der Datenbank gespeichert. Einem Alarm wird neben dem betroffenen User und dem entsprechenden Logeintrag auch ein Typ zugeordnet. Der Typ dient dazu, zwischen verschiedenen Alarmen unterscheiden zu können.

Die folgende Abbildung 4.3 des Entity-Relationship-Modell zeigt, wie die einzelnen Tabellen der Datenbank und ihre Attribute zueinander in Beziehung stehen.

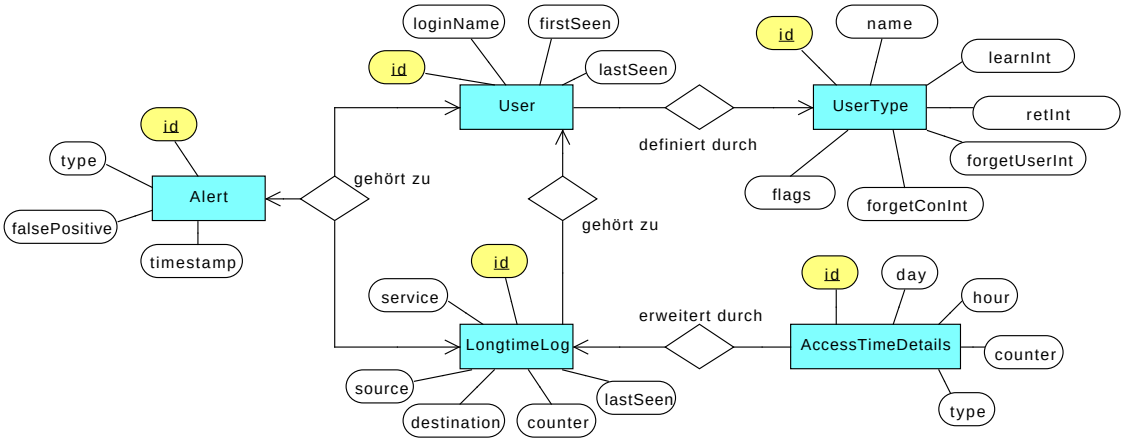

Abbildung 4.3.: Entity-Relationship-Modell des zentralen Datenbankschemas

Neben den bislang erwähnten Elementen wird in der Datenbank eine Tabelle zur Auflösung von IP-Adressen gespeichert. Dort können einzelne Netzblöcke auf einen Namen abgebildet werden. Dazu kann wahlweise eine Auflösung über Reverse-DNS oder ein Abgleich von regulären Ausdrücken der fraglichen IP-Adresse verwendet werden. Die Konfiguration erfolgt über einen Eintrag in der Tabelle `NetworkConfiguration`. Die Felder `useDNS`, `sourceSpec`, `doMapping` und `mappingResult` regeln hierbei das Verhalten. Da diese Funktionalität schon im Prototypen enthalten ist, wurde sie schon in [vEMH12b] vorgestellt. Die Funktionalität wird im Abschnitt 4.4.2 noch einmal beschrieben und die nötigen Änderungen aufgezeigt.

Ebenfalls wird die Datenbank zum Speichern von aus Logdateien gewonnenen Logzeilen und deren Inhalt verwendet. Dies geschieht in der Tabelle `LogEntries`. Wie in dem Abschnitt zum Lesen von Logdateien in Kapitel 4.1.1 erläutert, bieten die Datenbanktransaktionen bei der Verarbeitung von Logzeilen Vorteile.

Details zur Implementierung der Datenbank werden in Kapitel 5.4 beschrieben. Dort sind in

der Abbildung 5.2 alle Tabellen und Relationen als UML-Diagramm abgebildet.

Das überarbeitete Datenbankschema übernimmt die Tabellenstruktur die in dem LoginIDS-Prototypen eingeführt wurde und erweitert sie durch die Tabellen Alerts, AlertTypes, LogEntries und UserType.

4.4. LoginIDS Analyse

4.4.1. Der Analysealgorithmus

Der Analysealgorithmus verarbeitet die einzelnen Logeinträge und trifft auf Basis der in der Datenbank gespeicherten Informationen die Entscheidung, wann ein Alarm ausgelöst werden soll und wann nicht. Der im Folgenden beschriebene Ablauf wird in der Abbildung 4.4 als Aktivitätsdiagramm wiedergegeben. Der Übersicht halber ist nicht mit abgebildet, dass nur NEW USER und FLAGGED gemeldet werden, wenn der Benutzer in der Lernphase ist. Die einzelnen Logzeilen, die von den Diensten generiert werden, werden wie in Kapitel 4.1 beschrieben, von den Konvertierungsprogrammen zuerst in die Datenbank Tabelle LogEntries geschrieben. Der Analysealgorithmus arbeitet dann die Einträge dieser Tabelle ab. Dabei werden die nach ihrem Zeitstempel ältesten Einträge zuerst bearbeitet. Bei der Bearbeitung der Logeinträge wird pro Logeintrag maximal ein Alarm gemeldet. Es ergibt im Normalfall keinen Sinn, mehrere Alarme zu melden. Der NEW USER Alarm impliziert einen FIRST LOGIN und, da noch keine Zeit als „normale“ Zeit gespeichert wurde, auch einen SUSPICIOUS LOGIN, die daher nicht gemeldet werden müssen.

Die erste Überprüfung ergibt, ob der in dem Logeintrag aufgezeichnete Loginname schon in der LoginIDS-Benutzertabelle gespeichert ist. Wenn dies nicht der Fall ist und der Login als erfolgreich gekennzeichnet wurde, wird ein neuer Eintrag angelegt und der Alarm NEW USER gemeldet. Nicht erfolgreiche Anmeldungen von bislang unbekannten Benutzern werden von LoginIDS ignoriert. Das beugt einem potentiellen Angriff auf LoginIDS vor, bei dem die Datenbank mit unsinnigen Benutzern gefüllt wird. Falls der Benutzer vorher unbekannt war, sind auch noch keine Verbindungen und Zeitdetails zu den Verbindungen aufgezeichnet worden, die dann, wie weiter oben schon beschrieben, ohne einen weiteren Alarm auszulösen, in die Datenbank eingefügt werden. Außerdem werden die lastSeen und eventuell auch firstSeen Zeitstempel aktualisiert.

Zur Überprüfung auf einen FIRST LOGIN Alarm, wird die Datenbanktabelle LongtimeLog nach der Dienst, Quell- und Ziel-Adressen Kombination für den aktuellen Benutzer durchsucht. Bei der Analyse der Quelladresse wird diese unter Umständen durch das Mapping in der NetworkConfiguration auf einen anderen String abgebildet. Wenn die Kombination noch nicht vorhanden ist, wird ein FIRST LOGIN Alarm ausgelöst. Ansonsten wird der Datenbankeintrag aktualisiert, indem der lastSeen Zeitstempel und der Zähler aktualisiert werden. Nach der Aktualisierung wird für den MANY CONNECTIONS Alarm überprüft, ob das Verhältnis der Anzahl der unterschiedlichen Verbindungen des Benutzers zu den insgesamt aufgezeichneten Verbindungen größer als 90% ist. Dieser Alarm kann zum Beispiel ausgelöst werden, wenn ein Benutzer keine Verbindung zweimal verwendet, da er sich jedes mal von

einer anderen Quelladresse, die nicht durch die `NetworkConfiguration` auf einen gemeinsamen Namen projiziert werden, aus verbindet. In diesem Fall sollte eine Möglichkeit gesucht werden, die vielen Quelladressen auf einen eindeutigen Namen abzubilden. Mehr Informationen zu der Abbildung von IP-Adressen mit der `NetworkConfiguration` gibt es in Kapitel 4.4.2. Um auf den `RARE CONNECTION` Alarm zu testen, wird an derselben Stelle überprüft, ob das Verhältnis des Wertes des Zählers der aktuellen Verbindung zu allen aufgezeichneten Verbindungen kleiner als 10% ist. Der Benutzer verwendet diese Verbindung, im Verhältnis zu den anderen Verbindungen, also nicht sehr häufig. Die Werte 90% für `MANY CONNECTION` und 10% für `RARE CONNECTION` sind so gewählt worden, dass in der Testumgebung nicht zu viele Alarme dieses Typs ausgelöst werden. In anderen Umgebungen, zum Beispiel in einer Umgebung mit zwei Diensten, von denen pro Benutzer einer häufig mehr als 1000 Verbindungen hat und ein zweiter mit in der Regel weniger als 100 Verbindungen, müssen diese Werte geändert werden, damit nicht jeder Login auf dem zweiten Dienst einen `RARE CONNECTION` Alarm auslöst.

Der Alarm `SUSPICIOUS LOGIN` wird ausgelöst, wenn in der Datenbanktabelle `AccessTime Details` kein Eintrag, der zu der aktuellen Verbindung gehört und in der gleichen Stunde stattgefunden hat, gespeichert ist. Auch hier wird zusätzlich ein Zähler erhöht, der angibt, wie oft zu dieser Zeit schon die Verbindung hergestellt wurde. Dieser Zähler wird aber nicht zum Generieren eines eigenständigen Alarms verwendet. In der manuellen Analyse eines Ereignisses kann dieser Wert aber unter Umständen helfen festzustellen, wie oft ein Zugriff stattgefunden hat.

Eine weitere Überprüfung stellt fest, ob der Benutzer in einer Benutzergruppe ist, die markiert ist, und deren Benutzer immer einen Alarm auslösen sollen. Wenn das der Fall ist, wird der `FLAGGED` Alarm ausgegeben.

Nach der Analyse eines Logeintrages wird dieser in der Datenbank als bearbeitet markiert. Zusätzlich wird die Datenbanktransaktion, die zu Beginn der Analyse des Logeintrages gestartet wurde, an dieser Stelle durch einen `Commit` in die Datenbank eingebracht. Durch die Verwendung von Datenbanktransaktionen, ist sichergestellt, dass ein als bearbeitet markierter Eintrag auch wirklich vollständig bearbeitet wurde. Falls das Programm während der Analyse aus irgendeinem Grund abbricht, wird ein `Rollback` ausgeführt und dadurch keine schon aktualisierten Zähler oder Zeitstempel in der Datenbank gespeichert. Dadurch wird die Konsistenz des Datenbestandes sichergestellt.

Wenn alle neuen Logeinträge bearbeitet und „abgehakt“ wurden, überprüft der Algorithmus, ob es Benutzer gibt, die schon so lange inaktiv waren, dass sie aus dem Datenbestand gelöscht werden sollen. Dazu wird für jede Benutzergruppe, wie sie in Kapitel 3.2.4 vorgestellt wurden, das `forgetUserInt` berechnet. Für alle Benutzer, deren `lastSeen`-Zeitstempel älter ist, als das Intervall erlaubt, wird ein `EXPIRED USER` Alarm gemeldet und sie werden aus der Datenbank gelöscht. Dieser Alarm hat eine besondere Stellung, da er ebenfalls nicht in der Datenbank gespeichert wird. Ein Alarm in der Datenbank ist immer einem Benutzer zugeordnet, und der wird in diesem Schritt gelöscht. Gemeldet wird der Alarm trotzdem, um zum Beispiel in einem Trigger-Skript eine Mail zu versenden oder anders auf den Alarm zu reagieren. Die Trigger-Skripte werden in Kapitel 4.9 ausführlicher beschrieben. Beim Löschen eines Benutzers werden auch alle mit ihm in Verbindung stehenden Datenbankeinträge aus

den Tabellen `LongtimeLog`, `AccessTimeDetails` und `Alerts` gelöscht.

Nach dem gleichen Prinzip werden zu alte `LongtimeLog` Einträge, also Aufzeichnungen über Verbindungen, gelöscht. Wenn eine Verbindung älter als das `forgetConInt` ist, wird sie und die dazugehörigen `AccessTimeDetails` gelöscht. Auch die Alarme, die jetzt keiner Verbindung mehr zugeordnet werden können, werden gelöscht. In diesem Fall wird allerdings kein Alarm ausgelöst. Genau wie beim `EXPIRED USER` Alarm gibt es keine referenzierbare Verbindung mehr.

Dass an dieser Stelle doch viele Daten gelöscht werden, die letztendlich wegen fehlender Referenzen, nicht mehr für die automatisierte Analyse verwendet werden können, dient dem Datenschutz und soll einer massenhaften und dauerhaften Speicherung von Verbindungsdaten entgegenwirken. In Kapitel 4.5 wird dies ausführlich diskutiert.

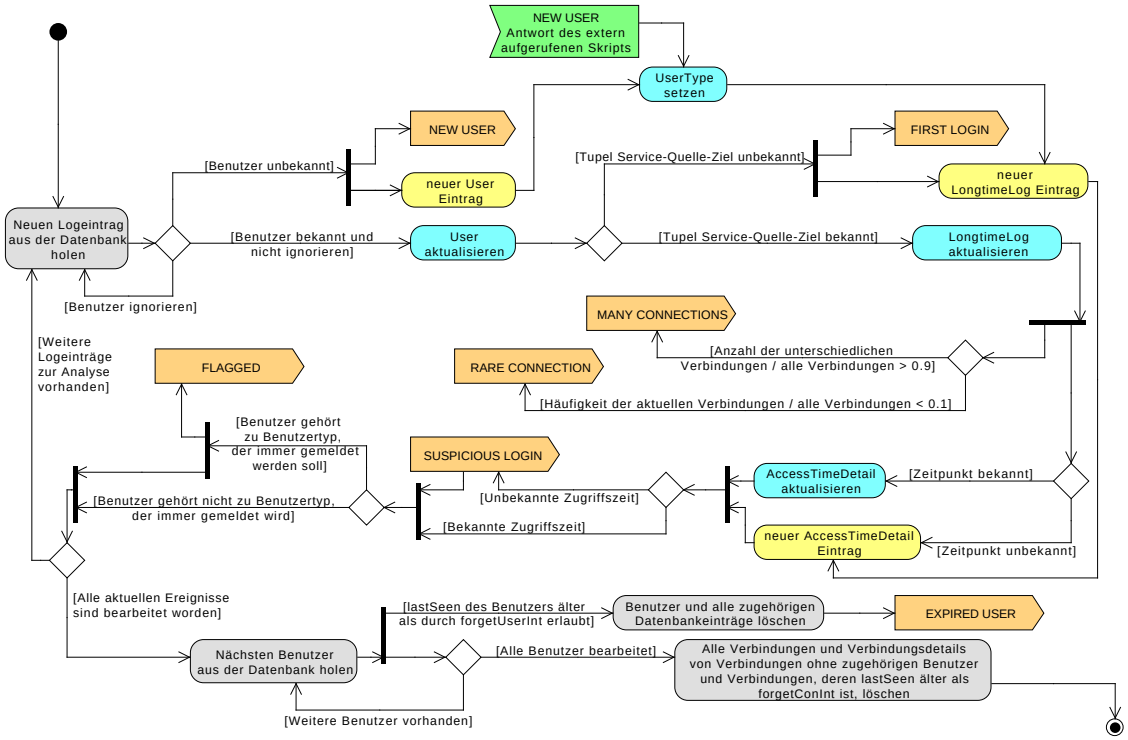

Abbildung 4.4.: Aktivitätsdiagramm des Analysezyklus von `loginidsd`

4.4.2. Funktion der `NetworkConfiguration`

Die `NetworkConfiguration` bietet die Möglichkeit je nach Quelladresse einen bestimmten anderen String in der Datenbank als Quelle zu speichern. Vor allem soll dies die Unterscheidung zwischen dynamisch und statisch vergebenen IP-Adressen ermöglichen. Der Bereich der dynamisch vergebenen Adressen kann so auf einen einfachen String abgebildet werden. Ansonsten würde jeder neue Login von einer dynamisch vergebenen Adresse einen `FIRST LOGIN` Alarm auslösen.

Das Feld `useDNS` gibt an, ob bei der Auflösung von IP-Adressen DNS verwendet werden soll. Gültige Werte sind hier 0 für „Nein“ und 1 für „Ja“. Wenn die DNS-Auflösung aktiviert ist, wird zunächst versucht die IP-Adresse aufzulösen. Um die Anforderung aus Abschnitt 3.2.12 zu erfüllen, muss hierzu noch die Funktion eines IPv6 Reverse-DNS Lookup implementiert werden. Danach wird das Ergebnis des Lookups oder die IP-Adresse selber mit der `SourceSpec` verglichen. Wenn dieser reguläre Ausdruck passt, wird die Quelladresse bei aktiviertem `doMapping` auf den String von `mappingResult` abgebildet. Wenn `doMapping` deaktiviert ist, wird die IP-Adresse auf den dazugehörigen DNS-Namen abgebildet. Diese Auswertung der einzelnen Felder wird in der Abbildung 4.5 als Aktivitätsdiagramm dargestellt.

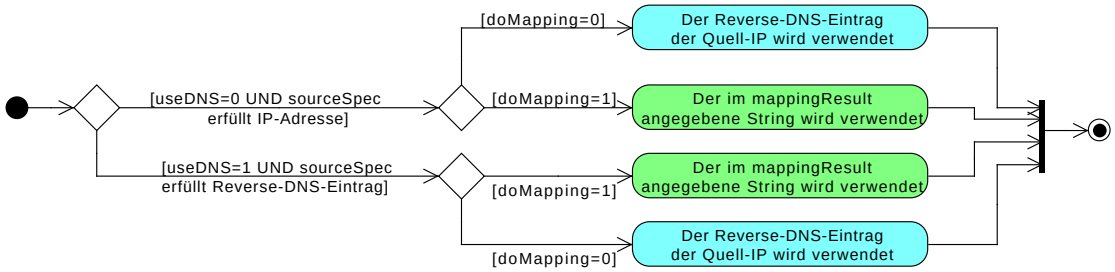

Abbildung 4.5.: Aktivitätsdiagramm der Auswertung von `NetworkConfiguration` Einträgen

Im Kapitel 5.5 wird die konkrete Implementierung dieser Funktion vorgestellt.

4.5. Retentionsintervall

Das Retentionsintervall bestimmt, nach welcher Zeitspanne Logdaten verworfen werden. Das Löschen von alten Logdateien wurde in der Anforderung aus Kapitel 3.2.3 beschrieben und ist in Deutschland durch den Gesetzgeber gefordert. Ausgehend von der Arbeit über Logfile-Management am LRZ [MHR11] ist die längste Zeit, die detailierte Logeinträge, aufgrund der aktuellen Rechtssprechung, gespeichert werden dürfen, sieben Tage. In `LoginIDS` kann dieser Zeitraum konfiguriert werden, standardmäßig wird aber eine Dauer von sieben Tagen empfohlen.

Das Retentionsintervall ist an mehreren Stellen von Belang. Zum einen werden Logdateien nach ihrer erfolgreichen Analyse in einen `done-`Ordner verschoben, aus dem sie gelöscht werden müssen. Genauso müssen die Logeinträge, die aus den Logdateien in die Datenbank übertragen werden, gelöscht werden. Dies geschieht nach dem in der Konfiguration festgelegten Intervall. Diese Informationen sind nur für `LoginIDS` während der Analyse und zu einer eventuellen Fehlerüberprüfung notwendig. Die Logdateien, die nicht analysiert werden konnten und einen Fehler verursacht haben, werden in einen `failed-`Ordner verschoben, der nicht automatisch bereinigt wird. Hier muss festgestellt werden, warum das Einlesen nicht funktioniert hat; zum Beispiel, weil durch ein Update eines Dienstes sich das Logformat geändert hat, und die Dateien, nach einer Anpassung des Konvertierungsskriptes, erneut in den `input-`Ordner kopiert werden.

Eine andere Stelle, an der das Retentionsintervall beachtet wird, ist, wann die Einträge des `DetailedLog` aus der Datenbank gelöscht werden. Diese sind nur zum manuellen Nachvollziehen einer Alarmmeldung gedacht Sie enthalten dazu alle gesammelten Informationen, werden aber für die Analyse zukünftiger Alarmmeldungen nicht verwendet.

Um in der Entwicklung reproduzierbare Ergebnisse zu haben, gibt es einen Debug-Mode, der jegliches Löschen von Daten abschaltet. Dazu kann das Retentionsintervall auf `-1` gesetzt werden. In einem Produktiveinsatz sollte dieser Modus aber unbedingt deaktiviert werden, um die gültigen Rechtsnormen einzuhalten.

Eine andere Stelle, an der Logeinträge zu löschen sind, ist das von `LoginIDS` zur Analyse verwendete `LongtimeLog`. Dort werden für die Benutzer die beobachteten Verbindungen gespeichert. Damit diese nicht unbestimmt lange gespeichert werden, werden Verbindungen, die innerhalb des Intervalls `forgetConInt` nicht mehr benutzt wurden, gelöscht. Dieses Intervall ist in der Standardeinstellung 28 Tage lang und kann über die Konfigurationsdatei und den Benutzertypen verändert werden. Wenn eine Verbindung aus dem `LongtimeLog` gelöscht wird, werden auch alle zugehörigen `AccessTimeDetails` und Alarmmeldungen, die mit dieser Verbindung verknüpft waren, gelöscht. Der Grenzwert von 4 Wochen sorgt dafür, dass sich nicht zu viele, kaum benutzte Verbindungen ansammeln und nach und nach jede Verbindung als „normal“ angesehen wird.

Nach einem ähnlichen Schema werden auch Benutzer aus der Datenbank entfernt. Wenn diese nicht innerhalb ihres `forgetUserInt`-Intervalls von `LoginIDS` gesehen wurden, werden sie aus der Datenbank gelöscht. In diesem Fall werden auch alle zugehörigen Verbindungen aus dem `LongtimeLog` gelöscht und in der Folge auch alle `AccessTimeDetails`-Einträge und Alarmmeldungen aus der `Alerts`-Tabelle. Das Standardintervall, nachdem ein Benutzer gelöscht wird, beträgt 84 Tage, was 12 Wochen entspricht. Diese Zeitintervall ist so gewählt, da ein regulärer Benutzer, der seinen Account 12 Wochen nicht mehr verwendet hat, ihn vermutlich nicht mehr benötigt.

Dadurch ist sichergestellt, dass ohne weitere Administration Benutzer, die keinen Zugriff mehr haben, automatisch „vergessen“ werden.

Die Implementierung der Retentionsintervalle wird in Kapitel 5.8 beschrieben.

4.6. Datenübertragung

In der Anforderung in Kapitel 3.2.5 wurde beschrieben, wie Logdateien auf den LoginIDS-Host übertragen werden können. Dazu werden hier die Möglichkeiten von SCP und rsyslog genauer betrachtet. Prinzipiell sind auch andere Methoden denkbar, solange sichergestellt ist, dass die Logdateien geeignet in den input-Ordner kopiert werden. Die spezielle Umsetzung für diese Arbeit am LRZ wird in Kapitel 5.2 vorgestellt.

4.6.1. SSH/SCP

Diese Methode zu verwenden, ist gerade auf Unix-Betriebssystemen mit sehr geringem Konfigurationsaufwand umsetzbar, da SSH/SCP meist vorinstalliert ist. Bei einer klassischen Linux-Dateisystem-Rechte-Verwaltung, kann für jeden Dienst, der seine Logfiles via SCP abliefern möchte, ein eigener Benutzer auf dem LoginIDS-Host erstellt werden. Falls die Unterscheidung zwischen den verschiedenen Dienstbenutzern nicht wichtig ist, könnte auch nur ein Benutzer angelegt werden und für jeden der Dienstbetreiber ein anderer Public-Key zur Authentifizierung hinterlegt werden. Alle diese Benutzer werden dann einer eigenen Gruppe zugeordnet, die auf dem LoginIDS-Input-Ordner write- und execute-Rechte bekommt. Das ist das Minimum an Rechten, die nötig sind, um eine Datei in den Ordner kopieren zu dürfen. Es ist mit dieser Methode aber leider nicht möglich ganz zu verhindern, dass auch Dateien in die andere Richtung, also vom LoginIDS-Host herunter, kopiert werden können. Die Gruppe sollte keine read-Rechte auf dem Ordner halten, damit der Inhalt des Ordners nicht angezeigt oder Wildcards, zum Herauskopieren, verwendet werden können. Trotzdem kann bei Kenntnis des genauen Dateinamens die Datei kopiert werden. Da das Schema der Dateinamen relativ strikt gegeben ist, muss angenommen werden, dass es mit nur wenigen Versuchen möglich ist, Dateinamen anderer Logfiles zu erraten. Das Format, dem die Logdateinamen entsprechen müssen, wurde in Kapitel 4.1 beschrieben. Durch die Implementierung der Real-Time-Auswertung, bei der der Input-Ordner von LoginIDS ständig überwacht wird, ist ein solcher Angriff nur sehr eingeschränkt über eine Race-Condition möglich. LoginIDS erkennt über Inotify wenn der Prozess, der eine Datei in den Input-Ordner kopiert, sein Filehandle schließt und damit offensichtlich fertig mit dem Kopiervorgang ist. Daraufhin wird die Datei sofort in einen working-Ordner verschoben, auf den niemand außer LoginIDS Zugriff haben muss. Um also Logdateien aus dem input-Ordner heraus kopieren zu können, muss der Angreifer in dieser kurzen Zeitspanne auf die Datei zugreifen. Ohne weitere Tests gemacht zu haben, vermute ich, dass die Chancen auf Erfolg sehr gering sind, vor allem, wenn die SSH-Zugriffe, zum Beispiel durch eine iptables-Regel, auf einmal pro Minute begrenzt sind. Im Zweifelsfall kann der Logdatei eine zufällige Dateiendung gegeben werden. Unter Einhaltung des generellen Dateischemas kann nach dem Punkt eine beliebige Zeichenkette stehen, solange der Dateiname gültig ist und das in Kapitel 4.1 beschrieben Schema eingehalten wird. Im Kapitel 5.2 wird eine mögliche Implementierung dieser zufälligen Dateiendung beschrieben.

Ein weiterer denkbare Fall, dass LoginIDS aus einem Grund momentan nicht auf dem LoginIDS-Host läuft und daher die Logdateien nicht aus dem input-Ordner entfernen kann,

kann dadurch abgedeckt werden, dass `LoginIDS` der Dienst-Gruppe beim Start `wx`-Rechte auf dem `input`-Ordner gibt und diese beim Beenden wieder entzieht. Dadurch können Dienste gar nicht auf den Ordner zugreifen während `LoginIDS` nicht läuft. Das erfordert eine gewisse Fehlerbehandlung auf dem Dienst-Host, der bei einer fehlgeschlagenen Übermittlung die Übertragung später noch einmal versuchen muss. So eine Fehlerbehandlung wäre aber in jedem Fall sinnvoll, da auch andere Software- oder Hardwarefehler eine Übertragung scheitern lassen können.

Des Weiteren muss überlegt werden, wie häufig Logdateien mit `SCP` übertragen werden. `LoginIDS` führt eine Differenzerkennung durch, sodass ohne Probleme das gleiche Logfile mehrmals zum `LoginIDS`-Host kopiert werden kann, ohne dass die ersten Einträge doppelt analysiert werden. Allerdings ist es je nach Netzauslastung nicht sinnvoll, das komplette Logfile minütlich zu kopieren, ohne genau zu wissen, ob überhaupt neue Logeinträge geschrieben wurden. Die `SSH/SCP` Methode ist daher nicht für eine Echtzeitauswertung geeignet. Je nach Größe des Netzwerkes und der Logdateien würde eine stündliche bis tägliche Übertragung die Netzlast in Grenzen halten.

4.6.2. `rsyslog`

Nachdem in der `SSH/SCP` Methode, die im vorherigen Abschnitt betrachtet wurde, einige Schwachpunkte aufgefallen sind, wird hier die Übertragung von Logdaten mit `rsyslog` betrachtet. Die Projekthomepage von `rsyslog` ist `http://www.rsyslog.com`. In vielen aktuellen Linux-Distributionen, zum Beispiel bei Debian, ist `rsyslog` der Standard syslog-Daemon oder er lässt sich, wie bei SUSE, einfach über den jeweiligen Paketmanager installieren [rsy09]. Mit `rsyslog` lassen sich syslog Ereignisse über das Netzwerk auf einem anderen Host loggen. `rsyslog` unterstützt dabei sowohl UDP als auch die verlässlichere Übertragung per TCP. Ebenso kann, um Vertraulichkeit der Logdaten zu gewährleisten, die Übertragung per GSSAPI oder TLS verschlüsselt werden. Da nur dann Daten übertragen werden, wenn auch ein neues Logereignis übermittelt werden soll, ist die Methode im allgemeinen sparsamer mit Netzwerkressourcen als die Methode über `SSH`.

Damit `rsyslog` richtig arbeiten kann, sollte die Datei, in die `rsyslog` die Logeinträge schreibt, nicht im `input`-Ordner von `LoginIDS` liegen. `LoginIDS` entfernt die Datei dort regelmäßig, wenn sie abgearbeitet wird. Das stört die Funktion von `rsyslog`. Besser ist es, zum Beispiel das Logfile von `rsyslog` über einen Cronjob, im Abstand von mehreren Minuten in den `input`-Ordner zu kopieren. Dabei werden prinzipiell wieder zu viele Logeinträge kopiert, da die Dateien sich aber auf dem selben Host befinden ist der I/O-Aufwand nicht so hoch und störend, als wenn die Datei ständig über das Netzwerk kopiert werden würde. Wie häufig das Logfile kopiert werden muss, ist abhängig davon, wie oft die Ausgabe von `LoginIDS` ausgewertet wird.

Prinzipiell ist auch die Umsetzung mit `syslog-ng` denkbar, welches eine Alternative zu `rsyslog` ist.

4.7. Setup-Skript

Zum einfachen Abfragen der initialen Konfiguration wird von `LoginIDS` ein Setup-Skript verwendet, das in einer interaktiven Shell Grundeinstellungen abfragt. Das Skript basiert auf Perl, damit die Datenbankanbindung und die benötigten Module direkt getestet werden können. Das Skript speichert die Eingaben in die Konfigurationsdatei `loginids.conf`.

4.8. Konfigurationsdatei

Die Konfigurationsdatei `loginids.conf`, die sich im Hauptverzeichnis von `LoginIDS` befindet, enthält alle wichtigen Einstellungen. Dort wird gespeichert, welche Datenbank verwendet werden soll. Das beinhaltet je nach verwendeter Datenbank und Konfiguration auch die Angabe von Username und Passwort. Die Datei sollte daher nur von `LoginIDS` lesbar sein. Nach Änderungen in der Datei müssen alle `LoginIDS`-Prozesse neu gestartet werden, damit die Änderungen übernommen werden. Die Datei `loginids.default.conf` enthält die Standardeinstellungen und dazu eine kurze Erklärung.

Weitere Konfigurationseinträge sind im Folgenden aufgelistet:

- **db_ds**: Dies ist der String, der von dem Perl-DBI-Modul verwendet wird, um eine Verbindung zur Datenbank herzustellen. Dieses Feld muss definiert werden.
- **db_user**: Hier kann der Benutzername, der bei der Verbindung zur Datenbank verwendet werden soll, angegeben werden.
- **db_password**: Analog kann hier das Passwort für den Datenbank-Benutzer angegeben werden.
- **loglevel**: Gibt an wie ausführlich das von `LoginIDS` geschriebene Logfile sein soll. Es werden alle Logmeldungen mit einem Level, das kleiner oder gleich dem hier angegebenen Loglevel ist, ausgegeben. Als Anhaltspunkt kann dabei gelten: Meldungen mit Loglevel 1 – 2 sind schwere Felder, 3 – 4 sind Warnmeldungen, 5 – 7 sind Informationsmeldungen und 8 – 9 sind Debugmeldungen. Wenn kein Loglevel angegeben wird, wird 5 als Standard verwendet.
- **retInt**: Das Retentionsintervall in Tagen. Wenn nicht anders angegeben, ist der Standardwert 7 Tage. Für den Wert 0 werden Logdateien und Detailinformationen sofort nach der Analyse gelöscht. Wenn der Wert gleich `-1` ist, werden Logdateien oder Detailinformationen nicht automatisch gelöscht.
- **learnInt**: Gibt das Lernintervall an, dass für neue Benutzer verwendet wird. Über den Benutzertyp kann dieses Intervall modifiziert werden. Der Standardwert ist 28 Tage.
- **forgetUserInt**: Dieses Intervall gibt an, nach welcher Zeit der Inaktivität ein Benutzer aus der Datenbank gelöscht wird. Der Standardwert ist 84 Tage, also nach ungefähr 3 Monaten.

- **forgetConInt**: Dieses Intervall gibt an, nach welcher Zeit eine zuvor aufgezeichnete Verbindung gelöscht werden soll. Der Standardwert ist 28 Tage.

4.9. Externe Skripte

Wie in der Anforderung in Kapitel 3.2.18 beschrieben, wird durch die Einführung von externen, durch Alarme auslösbare Skripte die Reaktion auf Alarme individuell anpassbar. Bei jeder Alarmmeldung versucht `LoginIDS` das Skript `triggers/triggers.pl` aufzurufen. Als Parameter wird dabei der aktuelle Alarm und die Alarmmeldung übergeben. In diesem Skript können dann weitere Aktionen ausgeführt werden. Grundsätzlich ist das `triggers.pl`-Skript nur als Wrapper zum alarmspezifischen Aufrufen weiterer Skripte gedacht. Dadurch wird die Komplexität der einzelnen Skripte gering gehalten und Veränderungen erleichtert. Der unter Umständen komplexe Programmcode kann in ein eigenes Skript ausgelagert werden. Die Abbildung 4.6 zeigt, wie der Ablauf für bestimmte Alarme aussehen könnte. Die Benutzervalidierung wird ebenfalls über diese externen Skripte ausgeführt. Dies wird im nächsten Kapitel 4.11 beschrieben.

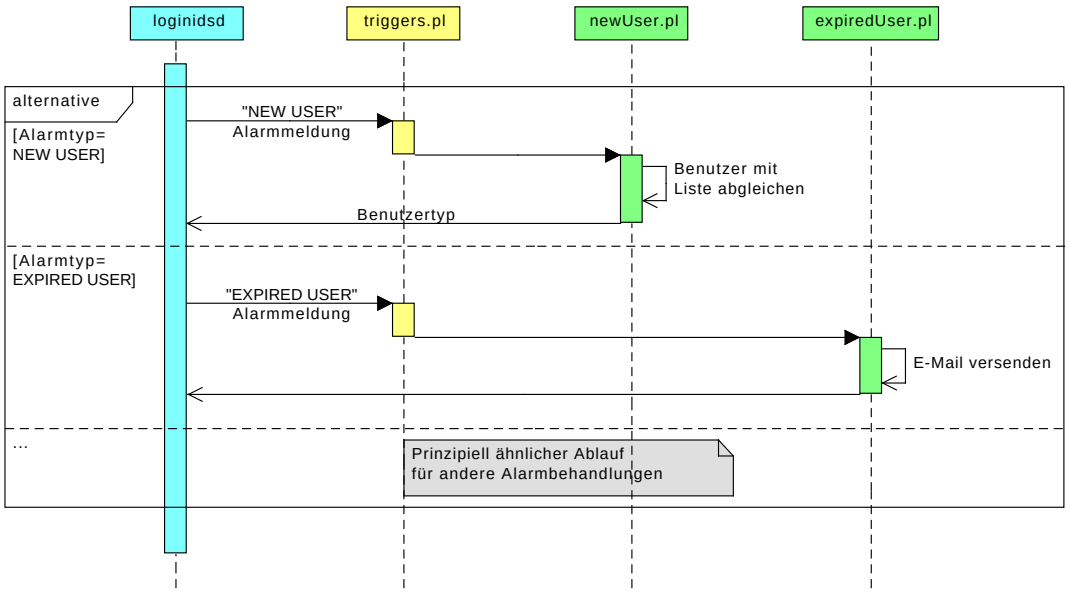

Abbildung 4.6.: Sequenzdiagramm einer möglichen Alarmbehandlung in dem Trigger-Skript

Wie Abbildung 4.6 zeigt, ist vorgesehen, dass das Trigger-Skript entscheidet, welches weitere Skript aufzurufen ist und dann den eigenen Prozess durch den des ausgewählten Prozesses ersetzt. Dies kann durch die Funktionen der `exec`-Familie geschehen und hat den Vorteil, dass der Standardoutput der ausgewählten Skripte ohne Umwege über das Trigger-Skript an `LoginIDS` gesendet und dort ausgewertet werden kann. Die Implementierung dieser Funktion wird in Kapitel 5.10 beschrieben.

4.10. Benutzertypen

Aus den Überlegungen aus Kapitel 3.2.4 heraus werden folgende Benutzertypen vorgeschlagen:

- **Default**: Standardeinstellung für den Benutzer verwenden.
- **Ignore**: Diesen Nutzer vollständig ignorieren.
- **Always**: Keine Lernphase für diesen Nutzer, jeder Login wird als Alarm gemeldet.
- **Short Learnphase**: 50% kürzere Lernphase.
- **Long Learnphase**: 50% längere Lernphase.
- **High Risk**: Benutzer mit besonderen Vereinbarungen, die eine längere Speicherung von detaillierten Informationen erlauben.

In der Implementierung am LRZ wird für alle LRZ-Mitarbeiter der `Default`-Benutzertyp mit 28 Tagen Lernphase verwendet. Alle Studenten, die auch Mitarbeiter sind und dadurch für kurze Zeit Zugang zu mehr Systemen haben, bekommen über den `Short Lernphase`-Benutzertyp nur eine Lernphase von 14 Tage. Alle anderen Studenten, haben eine längere Lernphase.

Wie diese Benutzertypen in das Datenbankschema integriert werden, wurde in Abschnitt 4.3 beschrieben. Außerdem wird die Möglichkeit, den Benutzertyp automatisiert festzulegen, überprüft. Am LRZ gibt es eine zentrale Benutzerverwaltung, die über LDAP geregelt ist. Dort sind die Benutzer in verschiedene Gruppen eingeordnet. Bei der Erkennung eines neuen Benutzers könnte der von `LoginIDS` verwendete Benutzertyp über die Gruppen, denen der Benutzer in dem LDAP-Verzeichnis zugeordnet ist, automatisch gesetzt werden. Diese Funktionalität wird bei Alarmen im Zusammenhang mit extern ausführbaren Skripten, wie sie in Kapitel 3.2.18 beschrieben wurden, untersucht. Die konkrete Implementierung und einige Besonderheiten der Benutzertypen werden in Kapitel 5.6 dargestellt.

4.11. Benutzervalidierung

Im Fall des LRZ wird hier der Benutzername des neuen Benutzers in einer Liste, die alle aktuellen im LDAP-Verzeichnis registrierten Benutzer enthält, gesucht und abhängig von den dort gesetzten Gruppenmitgliedschaften der Benutzertyp bestimmt.

Der Erkennungsmechanismus wurde, wie in Kapitel 4.4.1 beschrieben, so verändert, dass er nicht erfolgreiche Logins von bislang unbekannten Benutzern ignoriert. Daher würden die meisten Tippfehler ignoriert werden. Falls zufälligerweise ein gültiger Benutzername eingegeben wurde, kann aus dem Logfile, solange dort nicht das eingegebene Passwort steht, nicht überprüft werden, ob dies ein Fehler im Benutzernamen oder im Passwort war. Aus diesen Gründen wird diese Art der Validierung von Benutzernamen hier nicht implementiert.

Mit der Einführung von externen Skripten, wie sie in der Anforderung in Kapitel 3.2.18 beschrieben wurden, zum automatischen Setzen des Benutzertyps mit Informationen aus einem LDAP-Verzeichnis, einer Liste oder Ähnlichem, ist eine Benutzervalidierung beziehungsweise das Setzen eines bestimmen Benutzertyps für unbekannte Benutzer möglich.

4.12. Zeitfenster

In Kapitel 3.2.8 wurde vorgeschlagen die Speicherung der Login-Zeitpunkte zu verbessern. Dazu wäre es denkbar vier Sektionen zu speichern, 53–7, 8–22, 23–37 und 38–52. Damit würden 14:59 Uhr und 15:00 Uhr in dieselbe Sektion fallen und trotz unterschiedlicher Stundenwerte als „gleiche“ Zeit interpretiert. Sollte der Bereich von 14 Minuten zu eng sein, könnten auch noch angrenzende Sektoren akzeptiert werden. Bei einer näheren Untersuchung anhand von Abbildung 4.7, die zeigt, zu welcher Zeit meisten Logins stattfinden, wird klar, dass sich eine Lösung mit zwei Sektoren besser eignet. Durch die Minima im Graphen begrenzt, ergäben sich die Sektionen 13–42 und 43–12.

Da in beiden Fällen ähnlich viele Grenzfälle auftreten, wird die momentane Praxis, die Minuten abzuschneiden, beibehalten.

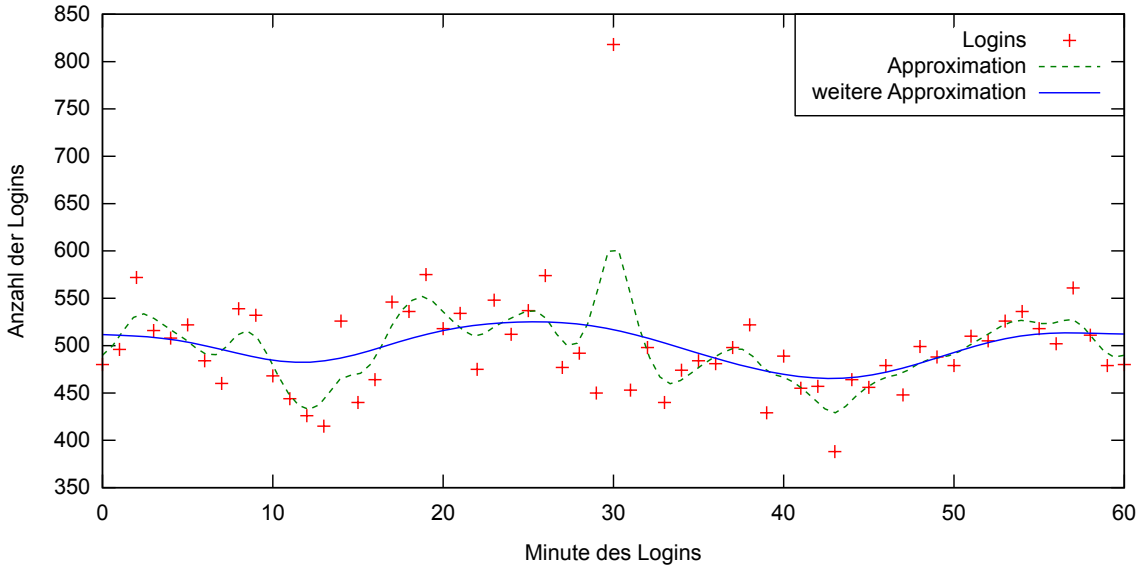

Abbildung 4.7.: Anzahl der Logins in Abhängigkeit von der Minute

4.13. Erweiterte Lernphase

Wie in der Anforderung von Kapitel 3.2.11 beschrieben, wird hier untersucht, welche Kriterien zum Beenden der Lernphase sinnvoll sind. Es wird überlegt, ob eine Kombination aus Zeitintervall und Mindestanzahl an stattgefundenen Logins die Zahl der Fehlalarme verringern kann.

So könnte die Lernphase nach einem Monat beendet werden, wenn mindestens 25 Loginvorgänge für diesen Nutzer beobachtet wurden. Im Testbetrieb wurden im ersten Monat durchschnittlich 27 ± 59 Logins pro Benutzer registriert. Das heißt, es werden durchschnittlich 27 Logins gezählt und die Standardabweichung, eine Gleichverteilung der Logins angenommenen, beträgt 59. Die hohe Abweichung ergibt sich durch eine sehr unterschiedliche Intensität der Nutzung von den mit dem Testsystem überwachten Systemen. Abbildung 4.8 zeigte wie oft welche Anzahl an Logins innerhalb des ersten Monats vorkommt. So wurde 13 mal nur ein und 20 mal zwei Logins festgestellt. Das andere Extrem mit etwas mehr als 600 Logins wurde nur einmal beobachtet. Da die Standardabweichung hier so groß ist, wird es keine Mindestanzahl für Logins geben. Die Dauer der Lernphase wird aber durch einen Wert in der Konfigurationsdatei frei konfigurierbar sein. Zudem könnte für die in Abschnitt 3.2.4 erwähnten verschiedenen Nutzertypen unterschiedliche Grenzwerte angegeben werden. Spätestens nach zwei Monaten sollte die Lernphase in jedem Fall beendet werden, da für ein Semester angestellte Mitarbeiter nur vier bis sechs Monate aktiv sind.

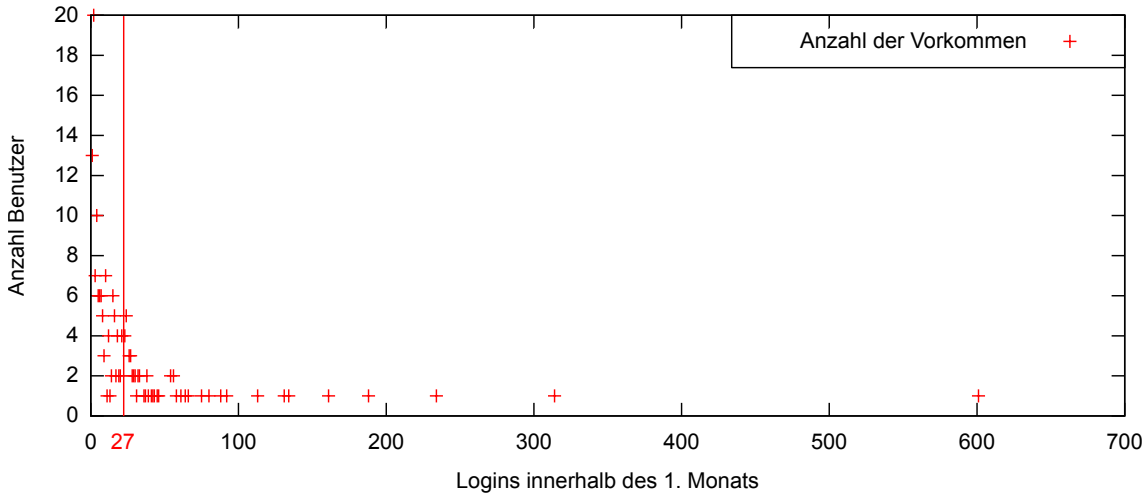

Abbildung 4.8.: Anzahl der Logins im ersten Monat

5. Implementierung

In diesem Kapitel wird die Implementierung von LoginIDS beschrieben. Implementiert ist LoginIDS in mehreren Teilprogrammen, die in Perl geschrieben sind. In dem Kapitel 5.1 wird die Struktur der einzelnen Programmteile und der Aufbau des Source-Code-Verzeichnisses dargestellt. Der Source-Code ist auf https://git.lrz.de/?p=LoginIDS.git;a=summary unter der GPLv3 Lizenz veröffentlicht. Zusätzlich ist das Projekt auf https://freecode.com/projects/loginids zu finden.

5.1. Struktur der LoginIDS-Programme

LoginIDS besteht aus mehreren Perl-Skripten, die gemeinsam die Funktionalität sicher stellen. Die beiden Hauptskripte, loginidsd.pl und logindexd.pl, sind so konzipiert, dass sie als Daemon ausgeführt werden und ständig laufen. Im Folgenden wird bei diesen Skripten die Dateiendung in den meisten Fällen weggelassen. Gemeinsam genutzte Programmteile sind in dem Perlmodul LoginIDS.pm ausgelagert. Dieses Modul enthält die Methoden zum Lesen der Konfigurationsdatei, zum Öffnen und Schreiben der Logfiles und der Verwendung von Lockfiles. Da loginidsd und logindexd als Daemon konzipiert sind, darf jeweils nur eine Instanz von ihnen laufen. Dies wird über Lockfiles sichergestellt. Zum Starten von LoginIDS existiert das Shell-Skript loginids, das mit den Parametern „start“ und „stop“ aufgerufen werden kann, um die LoginIDS-Programme zu starten und zu beenden.

Die wichtigste Konfiguration von LoginIDS erfolgt durch die Datei loginids.conf. Diese Datei wird auch automatisch beschrieben, wenn das setup.pl Skript ausgeführt wird, das beim Anwender die wichtigsten Einstellungen abfragt.

Neben diesen zentralen Bestandteilen von LoginIDS existieren noch weitere Skripte, die die Konvertierung von Logdateien übernehmen. Diese befinden sich in dem Ordner converter-available und werden, wenn sie aktiviert werden sollen, über einen Link in dem Ordner converter-enabled dem ihrem Namen entsprechenden Logfile zugeordnet.

Weitere Skripte, die individuell angepasst werden können, befinden sich in dem Ordner triggers. Bei allen Alarmmeldungen wird dort das Skript triggers.pl aufgerufen, falls es ausführbar ist.

Die Abbildung 5.1 gibt einen Überblick über die Struktur des Source-Verzeichnisses inklusive der Angabe über die Anzahl der Code-Zeilen (Lines of Code) für jede Source-Datei.

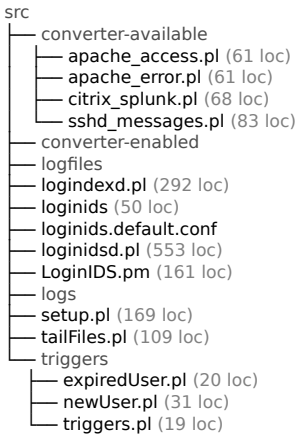

Abbildung 5.1.: Übersicht über die Dateien des src-Verzeichnisses mit Angabe der Anzahl der Codezeilen

5.2. Übertragen von Logfiles

Die verschiedenen Methoden zum Übertragen von Logfiles wurden in Kapitel 4.6 beschrieben. In diesem Kapitel wird die spezielle Umsetung in dem `LoginIDS`-Testbetrieb beschrieben.

Die zu analysierenden `SSH`-Server-Logdateien werden über `rsyslog` auf dem Testsystem in `/var/log/wsc.log` gespeichert. Ursprünglich wurde die Datei von dort einmal am Tag in den `input`-Ordner von `LoginIDS` verschoben. Mit der Einführung der Real-Time-Analyse-Fähigkeit mit `logindexd` und `loginidsd`, wird dieses cronjobgesteuerte Verschieben des Logfiles zuerst in ein 5 Minuten Intervall verkürzt. Mit dem `tailFiles`-Programm ist dies nicht mehr nötig, da die Logdatei direkt überwacht wird.

Die Logdateien der Citrix-Windows-Terminalserver werden aus einer `Splunk`-Suche extrahiert und deren Inhalt in eine Logdatei geschrieben. Diese wird dann per `SSH/SCP` an den `LoginIDS`-Host geschickt wird. Dies geschieht nur einmal am Tag.

Die Frequenz, mit der die Logdateien an `LoginIDS` gesendet werden, ist letztendlich davon abhängig, wie oft die Ausgabe überprüft wird. Für den Testbetrieb ist einmal täglich prinzipiell ausreichend.

In dem Kapitel 4.6.1 wurde für die Methode mit `SSH/SCP` vorgeschlagen eine zufällige Dateiendung zu erzeugen, um zu verhindern, dass andere Dienste, die auch Zugang zum `input`-Ordner haben, Dateien mit dem Wissen über den Dateinamen herauskopieren können. Unter Linux könnte ein pseudo Zufallswert von acht Zeichen mit dem Kommando „`date +%s | sha1sum | base64 | head -c 8`“ erstellt werden.

5.3. Einlesen von Logfiles

Die in Abschnitt 4.1 beschriebenen Funktionen werden durch den Daemon `logindexd` und die entsprechenden Konvertierungsskripte implementiert. `logindexd` ist als Daemon konzipiert, das heißt, er läuft prinzipiell dauernd und überwacht den `input`-Ordner. Wenn zum Zeitpunkt des Startes von `logindexd` schon Dateien im `input`-Ordner sind oder später Dateien hinzukommen, entscheidet `logindexd` mit welchem Konverterskript diese auszulesen sind und ruft das entsprechende Skript auf.

5.3.1. Vorverarbeitung der Logfiles in `logindexd`

Beim Start von `logindexd` wird nach dem Initialisieren, bei dem das Working-Directory geändert, Lock- und Log-Files geöffnet werden und ein Signal-Handler installiert wird, sofort der `input`-Ordner mit Inotify überwacht. Dieser Ablauf ist in Listing 5.1 dargestellt. Der Daemon `logindexd` erhält eine Nachricht, wenn eine neue Datei im `input`-Ordner abgelegt wird und der Filehandle auf der Datei geschlossen wird. Zusätzlich wird überprüft, ob sich schon Logdateien in dem `input`-Ordner befinden, die dort vor dem Start von `logindexd` abgelegt wurden. Die Dateinamen werden dann in ein Array eingelesen. Dort werden sie entsprechend ihres Zeitstempels sortiert und verarbeitet, wenn kein anderer Prozess Zugriff auf sie hat. Die Sortierungsfunktion wird in einem eigenen Abschnitt in Kapitel 5.3.2 beschrieben. Der Systemaufruf von `lsof` mit dem Dateinamen gibt einen Wert ungleich Null zurück, wenn kein Prozess auf die Datei zugreift. Bei der Funktion, die die Dateien einliest, ist zu beachten, dass nur reguläre Dateien eingelesen werden. Ignoriert werden insbesondere „versteckte“ Dateien deren Name mit einem Punkt beginnt.

Listing 5.1: Setup-Funktion von `logindexd`

```
1  chdir(dirname($0));
2  setupLock();
3  %config = readConfig();
4  setupLogfile();
5  logMsg(7, "Config read and logfile opened.");
6  logMsg(3, "Not deleting old logfiles: negative retInt") if($config{retInt} < 0);
7  $SIG{'INT'} = \&interruptHandler; # signal int
8  $SIG{'TERM'} = \&interruptHandler; # signal term
9  logMsg(7, "Signal handler installed.");
10 $inotify = Linux::Inotify2->new; # create new inotify
11 $inotify->watch($inputDir, IN_CLOSE_WRITE, \&handleNewFile); # setup inotify
12 logMsg(7, "Inotify set up.");
13 @initialFiles = (<$inputDir/*>); # get all files from the input directory
14 logMsg(7, "Found ". @initialFiles ." initial files.");
15 my @sortedFiles = sort{&compareFileTimestamps}@initialFiles;
16 foreach my $file (@sortedFiles) {
17     system("lsof " . $file . " > /dev/null"); # check for other processes
18     if($?==0) {
19         logMsg(4,"Not moving " .$file." because some other process is accessing it.")
20     } else {
21         logMsg(5,"(initial) " . $file);
22         processFile($file);
23     }
24 }
```

Nachdem die Initialisierung durchgeführt wurde, kehrt das Programm zurück in die Main-Schleife, wo es in einer Endlosschleife wartet, um entweder durch Inotify oder durch ein Signal aktiviert zu werden. Wie im vorherigen Listing 5.1 zu sehen war, wird bei einer Aktivierung durch Inotify die Funktion `handleNewFile()` aufgerufen. Diese Funktion ist in Listing 5.2 dargestellt. Da an dieser Stelle kein Sortieren mehr stattfindet und auch nicht gewartet wird, welche Datei eventuell als nächstes in den `input-`Ordner kopiert wird, ist es wichtig die Dateien in der zeitlich richtigen Reihenfolge in den `input-`Ordner zu kopieren. Beim Start von `logindexd` werden die im `input-`Ordner liegenden Dateien, wie in Kapitel 5.3.2 beschrieben, sortiert. Ebenfalls wichtig ist, dass die Dateien *kopiert* und *nicht verschoben* werden. Das Event, auf das Inotify wartet, ist `IN_CLOSE_WRITE`, welches bei einem einfachen Verschieben nicht ausgelöst wird. Die Datei würde dann erst beim nächsten Start von `logindexd` verarbeitet werden.

Listing 5.2: Inotify-Handler von `logindexd`

```
1 sub handleNewFile() {
2     my $file = shift;
3     logMsg(5, "(triggered) " . $file->fullname);
4     processFile($file->fullname);
5 }
```

Sowohl die Funktion `handleNewFile()` als auch die vorher in Listing 5.1 gezeigte `setup()`-Funktion rufen auf den zu verarbeitenden Dateien die Funktion `processFile()` auf. Diese Funktion übernimmt die Überprüfung, ob die Datei schon teilweise analysiert wurde und entscheidet welcher Konverter für die Datei verwendet wird.

Während der Bearbeitung durch `processFile()` wird die Logdatei aus dem `input-` in den `working-`Ordner verschoben. Wenn festgestellt wird, dass die Logdatei genauso beginnt wie die aktuellste Logdatei dieses Logtyps im `done-`Verzeichnis, werden die Zeilen, die zusätzlich in der neuen Logdatei sind, in eine eigene Datei im `parts-`Ordner geschrieben und diese Datei von dem Konvertierungsskript bearbeitet. Nach erfolgreicher Bearbeitung wird die Datei aus dem `working-`Ordner in das `done-`Verzeichnis verschoben.

5.3.2. Zeitliches Sortieren der Logdateien

Beim initialen Aufruf von `logindexd` müssen die in dem `input-`Ordner angesammelten Logdateien in der richtigen Reihenfolge abgearbeitet werden. Damit die Dateien sortiert werden können, muss zuerst der Zeitstempel aus dem Dateinamen extrahiert werden. Der relativ strikte Aufbau und die gute Unterstützung von regulären Ausdrücken in Perl erleichtern diese Aufgabe. In der Sortierfunktion `compareFileTimestamps` wird aus den speziellen Variablen `$a` und `$b`, die beim Aufruf der Funktion mit den zu vergleichenden Dateinamen belegt werden, ein Datum extrahiert.

Der reguläre Ausdruck sucht dabei nach den durch das Namensschema vorgegebenen Minuszeichen im Dateinamen und selektiert in der ersten Gruppe das aus Zahlen und Trennzeichen bestehende Datum. Nachdem das Datum geparsed wurde, kann es verglichen werden. Diese Aufgabe wird durch die Bibliotheksfunktion `DateTime->compare` ausgeführt. Mit Hilfe der

Rückgabewerte dieser Funktion, kann die `sort`-Funktion aus 5.1 das Array mit Dateinamen sortieren.

Listing 5.3: Auslesen und Vergleich des Zeitstempels aus dem Dateinamen

```
my ($date1, $date2);
if ($a =~ /[^-]*[-][^-]*[-]([\dT:-]*)(\..*)?$/) {
    $date1 = myParseDateTime($1);
}
if ($b =~ /[^-]*[-][^-]*[-]([\dT:-]*)(\..*)?$/) {
    $date2 = myParseDateTime($1);
}
my $returnVal =  DateTime->compare($date1, $date2);
return $returnVal;
```

5.3.3. Konvertierungsskripte

Mit Hilfe des sortierten Arrays können die Dateien nun verarbeitet werden. Dazu wird durch einen regulären Ausdruck festgestellt, welchen Typ eine Eingabedatei hat und dann das entsprechende Konvertierungsskript aufgerufen. Sollte bei der Konvertierung ein Fehler auftreten, wird die Datei in den Ordner `failed` verschoben. Der Grund des Fehlers muss dann anhand der Logausgabe ermittelt werden. Wenn kein Fehler auftritt, kann `loginidsd` die in die Datenbank übertragenen Logereignisse analysieren und die Logdatei wird von `logindexd` in den Ordner `done` verschoben. Der Codeauszug in Listing 5.4 zeigt diese Funktion.

Listing 5.4: Bestimmung des nötigen Konverterskriptes

```
if($filename =~ m/^([^-]+)[-]/) {
    my $converter = "./converter-enabled/".$1;
    logMsg(7, "(converting) " . $filename . " using converter: " . $converter);
    system("perl " . $converter . " " . $workingFile);
    if ( $? ) {
        logMsg(3, "(converting) returned error: $?");
        move($workingFile, $failedDir . "/") or print "move failed: $!\n";
        return;
    } else {
        logMsg(7, "(converting) finished successful");
    }

    if($isPartialFile){
        move($workingFile, $partDir . "/") or print "move failed: $!\n";
    } else {
        move($workingFile, $doneDir . "/") or print "move failed: $!\n";
    }
} else {
    logMsg(3, "(converting) " . $filename . " could not find converter.");
}
```

5.4. Datenbankschema

Das in Kapitel 4.3 entworfene Datenbankschema wird für eine `SQLite`- und `MySQL`-Datenbank umgesetzt. Die beiden SQL-Dialekte unterscheiden sich in einigen Befehlen, so dass bei der Implementierung an manchen Stellen, je nach verwendeter Datenbank, unterschiedliche Anfragen implementiert werden müssen. Solche Situationen betreffen vor allem das Erstellen der benötigten Tabellen und das Selektieren von Einträgen mit einem bestimmten Datum. Einfachere Anfragen wie die meisten anderen Selektionen, Einfüge und Update-Operationen können für beide Datenbanktypen verwendet werden. Die Übersetzung der Perl-Variablen in die entsprechenden Datenbanktypen wird von dem Perl-Modul DBI übernommen. DBI unterstützt sehr viele verschiedene Datenbanksysteme, so dass eine zukünftige Unterstützung für ein weiteres System nur an wenigen Stellen Änderungen erfordert.

Das Erstellen der nötigen Tabellen übernimmt das Setup-Skript `setup.pl`. Dieses Skript wurde in Kapitel 4.7 beschrieben und übernimmt die Abfrage der wichtigsten Einstellungen zum Anlegen und Konfigurieren der Datenbank. Innerhalb dieses Skriptes erfolgt das Erstellen der einzelnen Tabellen durch das DBI-Modul. Dadurch können Einfügeoperationen ebenfalls weitestgehend unabhängig von dem konkret verwendeten Datenbanksystem ausgeführt werden.

Das schon in Abbildung 4.3 skizzierte zentrale Datenbankschema wird in Abbildung 5.2 nun vollständig gezeigt. Die Abkürzungen hinter den Dateitypen markieren bestimmte Schlüsselattribute. `<<PK>>` bedeutet „Primary Key“, `<<AK>>` bedeutet „Alternative Key“ und `<<FK>>` steht für „Foreign Key“.

Einige Felder sind in der Datenbank vorgesehen und angelegt, werden aber von `LoginIDS` momentan noch nicht verwendet. Dazu gehört das Attribut `type` in der `AccessTimeDetails`-Tabelle, der dazu verwendet werden soll, anzuzeigen, ob ein Login oder Logout aufgezeichnet wurde. Diese Funktionalität wurde in Kapitel 4.1.3 vorgeschlagen, nach einer näheren Untersuchung aber nicht implementiert. Auch das Attribut `falsePositive` in der `Alerts`-Tabelle wird noch nicht verwendet, da es keinen „Rückkanal“ von `Splunk` in die `LoginIDS`-Datenbank gibt, durch den ein Administrator einen Alarm als „false positive“ markieren könnte.

5.5. `NetworkConfiguration`

Die `NetworkConfiguration` ist eine spezielle Tabelle in der `LoginIDS`-Datenbank. Hier können wie in Kapitel 4.4.2 beschrieben, Einstellungen zum Abbilden von IP-Adressen auf Namen gemacht werden.

5.5.1. Erweiterung der `NetworkConfiguration`

Mit dem Auftreten neuer IP-Adressen können die Mappingtabellen erweitert werden. Zum Beispiel um die von der Deutschen Telekom vergebenen dynamischen IP-Adressen abzubilden, wurde eine Regel hinzugefügt. Dadurch werden IP-Adressen, die mit Reverse-DNS zu

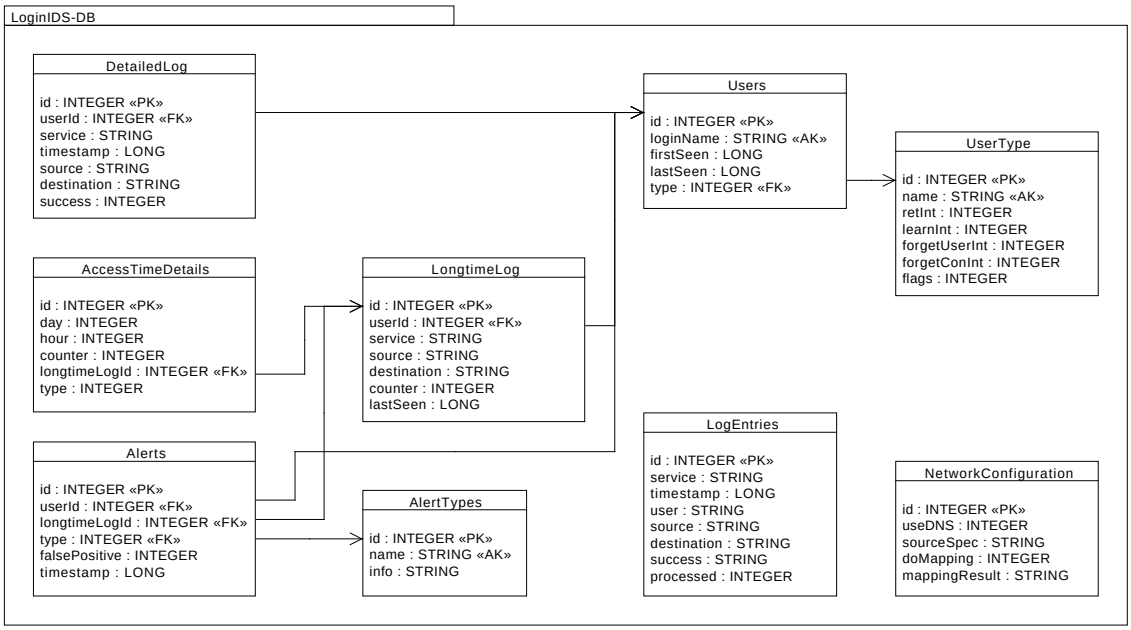

Abbildung 5.2.: UML-Diagramm der gesamten `LoginIDS`-Datenbankstruktur

`*.dip.t-dialin.net` aufgelöst werden können, auf die Zeichenkette „`Deutsche Telekom DIP`“ in der Datenbank abgebildet. Die folgende Tabelle zeigt noch weitere implementierte Abbildungsregeln.

useDNS	sourceSpec	doMapping	mappingResult
1	`.*\.t-dsl\.de`	1	Telekom-DSL
1	`.*\.lrz-muenchen\.de`	1	LRZ-Oldschool
0	`129\.187\.48\..*`	1	LRZ-MitArb-VPN
1	`.*\.dip\.t-dialin\.net`	1	Deutsche Telekom DIP
1	`.*\.pool\.mediaWays\.net`	1	Telefonica Deutschland
1	`.*\.web\.vodafone\.de`	1	Vodafone D2
1	`.*\.dynamic\.cablesurf\.de`	1	Kabelfernsehen Muenchen
1	`.*\.dynamic\.mnet-online\.de`	1	M-net Customer
1	`.*\.customer\.m-online\.net`	1	M-net Dynamic

Problematisch ist das nachträgliche Einfügen neuer Regeln. Wenn für einen Benutzer schon mehrere dynamisch zugewiesene IP-Adressen im `LongtimeLog` stehen, wird nach dem Aktivieren einer neuen Regel zunächst ein `FIRST LOGIN` für den gemappten Wert gemeldet. Eventuell könnten die alten Datenbankeinträge durch ein Skript überprüft und das Mapping nachträglich eingefügt werden. Diese Methode würde aber die Eindeutigkeit des Tupels, Service, Source und Destination verletzen. Daher müssen die Einträge, die in den drei Feldern identisch sind, zusammengefasst werden. Das hat dann Auswirkungen auf die Tabelle `AccessTimeDetails`, die einen Eintrag im `LongtimeLog` referenziert. Daher müssen auch diese Einträge zusammengefasst und aktualisiert werden. Der Aufbau der Datenbank und die

Abhängigkeiten der einzelnen Tabellen wurde in Abbildung 4.3 dargestellt. Da alte Verbindungsinformationen nach einem Retentionsintervall, wie in Kapitel 4.5 beschrieben, entfernt werden, ist es weniger fehleranfällig und komplex auf ein nachträgliches Zusammenlegen von Einträgen zu verzichten und dafür nach einer Änderung an der `NetworkConfiguration` für den geänderten Eintrag unter Umständen `FIRST LOGIN` Alarme zu bekommen, die dann ignoriert werden können.

5.5.2. DNS-Mapping

Zum Auflösen von IP-Adressen zu DNS-Namen, wird hier ein Reverse-DNS-Lookup verwendet. Die dazu nötigen Methoden werden von den Perl Modulen Socket und Socket6 bereitgestellt. Die Unterscheidung zwischen IPv4-Adresse und IPv6-Adresse wird über Patternmatching mit regulären Ausdrücken sichergestellt. Da bei IPv6-Adressen sehr viele verschiedene Schreibweisen erlaubt sind, ist der reguläre Ausdruck, der auf IPv6-Adressen prüft, nicht vollständig. Er erlaubt auch viele Zeichenketten, die keine gültigen IPv6-Adressen sind. Um die Komplexität des regulären Ausdruckes aber nicht deutlich zu erhöhen, oder neue Perl Module zu benötigen, die eine vollständige Validierung durchführen können, wurde dieser Ausdruck beibehalten. Alle gültigen IPv4-Adressen werden durch die erste Überprüfung verarbeitet, so dass bei der Überprüfung auf IPv6-Adressen nur noch IPv6-Adressen oder im Fehlerfall ungültige Strings auftreten können. Damit nicht ständig DNS-Anfragen gestellt werden müssen, wird ein lokaler Cache aufgebaut, der alle schon einmal gesuchten Einträge speichert. Dieser Cache wird einmal am Tag gelöscht.

Listing 5.5: Reverse-DNS-Auflösung

```
1  sub getDNSname($) {
2      my $ipaddr = shift;
3
4      return $cachedDNS{$ipaddr} if (defined($cachedDNS{$ipaddr}));
5
6      my $name;
7      if($ipaddr =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/) { # IPv4
8          $name  = gethostbyaddr(inet_aton($ipaddr), AF_INET);
9      } elsif($ipaddr =~ /(.*):(.*):[a-f0-9]+/i) { # something that looks like an IPv6
10         my $ip6_packed = inet_pton(AF_INET6, $ipaddr);
11         my $sockaddr = pack_sockaddr_in6(80, $ip6_packed);
12         ($name) = getnameinfo($sockaddr);
13     } else {
14         logMsg(9, "getDNSname: $ipaddr is not a valid IP address");
15         $cachedDNS{$ipaddr} = "invalid";
16         return $cachedDNS{$ipaddr};
17     }
18     unless($name) {
19         $cachedDNS{$ipaddr} = "unknown";
20     } else {
21         $cachedDNS{$ipaddr} = $name;
22     }
23     return $cachedDNS{$ipaddr};
24 }
```

5.6. Benutzertyp

In Kapitel 4.10 wurden einige mögliche Benutzertypen vorgestellt. Die Abbildung 5.2 zeigt wie die Benutzertypen in das Datenbankschema integriert werden. In diesem Kapitel wird erklärt, was die einzelnen Attribute zu bedeuten haben.

Die Werte `retInt`, `learnInt`, `forgetUserInt` und `forgetConInt` entsprechen grundsätzlich den gleichnamigen Werten in der Konfigurationsdatei, wie sie in Kapitel 4.8 vorgestellt wurden. Innerhalb der Benutzertypen können die Werte aus der Konfigurationsdatei modifiziert oder überschrieben werden. Das genau Verhalten wird dabei über die `flags` gesteuert. Das Attribut `flags` ist in der Datenbank ein Integerwert, der seine Bedeutung durch eine Bit-Maske erhält. Die folgende Tabelle beschreibt die möglichen Optionen.

		Bedeutung wenn Bit	
Bit	Modifiziertes Attribut	gesetzt	nicht gesetzt
1	`retInt`	Wert ist Prozentangabe	Wert ist in Tagen
2	`learnInt`	Wert ist Prozentangabe	Wert ist in Tagen
4	`forgetUserInt`	Wert ist Prozentangabe	Wert ist in Tagen
8	`forgetConInt`	Wert ist Prozentangabe	Wert ist in Tagen
16	—	Immer einen Alarm auslösen	
32	—	Nie einen Alarm auslösen	

Wenn in `flags` also das Bit 1 gesetzt ist, wird der Wert, der bei `retInt` angegeben wurde, als Prozentangabe interpretiert. Im Beispiel des Benutzertypes mit reduzierter Lernphase sind in der Datenbank die folgenden Werte angegeben: `retInt` = 0, `learnInt`= -50, `forgetUserInt` = 0, `forgetConInt` = 0 und `flags` = 15. Das `flags` = 15 bedeutet, dass Bits 1,2,4 und 8 gesestzt sind und daher die Werte für die Intervalle als Prozentangaben interpretiert werden. Die Prozente werden als Änderung zu den in der Konfigurationsdatei oder Standardkonfiguration angegebenen Werten betrachtet. In dem aufgeführten Beispiel werden `retInt`, `forgetUserInt` und `forgetConInt` nicht verändert (Änderung um 0%) und von `learnInt` wird die Hälfte abgezogen.

Wenn der Wert als Anzahl von Tagen interpretiert wird, ist dies absolut. Also werden, wenn das Retentionsintervall auf `retInt` = 14 gesetzt ist und bei `flags` das Bit 1 nicht gesetzt ist, die Einträge dieser Benutzergruppe nach 14 Tagen gelöscht. Vollkommen unabhängig von dem in der Konfiguration angegeben Wert.

Wenn das Bit mit dem Wert 16 gesetzt ist, wird für Benutzer dieser Gruppe der Alarm `FLAGGED` ausgelöst, wenn kein anderer Alarm aufgetreten ist. Wenn das Bit mit Wert 32 gesetzt ist, wird für alle Benutzer dieser Gruppe kein Alarm ausgegeben. Ab Loglevel 7 wird aber vermerkt, dass ein Benutzer ignoriert wurde. Die Werte 16 und 32 gleichzeitig zu verwenden funktioniert nicht. Die Überprüfung auf Ignorieren erfolgt zuerst und entscheidet dadurch das Verhalten von `LoginIDS`.

5.7. Der Analysealgorithmus

Der Analysealgorithmus von `LoginIDS` wurde in Kapitel 4.4.1 beschrieben. In diesem Abschnitt werden einige ausgewählte Teile der Implementierung dargestellt.

Das Listing 5.6 zeigt die Hauptschleife, die `loginidsd` im Intervall von `loginidsdCheckInt` durchläuft. Dieser Wert kann in der Konfigurationsdatei angepasst werden und hat einen Standardwert von 30 Sekunden. In jedem Durchlauf wird die Verbindung zur Datenbank neu hergestellt. Dadurch können zufällige Verbindungsabbrüche und Probleme beim Locken von `SQLite` Datenbanken umgangen werden. Danach werden aus der `LogEntries`-Tabelle alle Einträge, die noch nicht mit `processed=0` abgehakt sind, aufgelistet. Die Einträge werden dann in der while-Schleife nacheinander abgearbeitet. Dabei wird jeder Eintrag mit einer eigenen Datenbank-Transaktion behandelt. Nach der erfolgreichen Verarbeitung eines Eintrags wird die Transaktion durch einen `Commit` abgeschlossen. Nachdem alle neuen Einträge verarbeitet wurden, wird in den Funktionen `removeOldDbLogEntries()` und `removeOldUsersAndConnections()` überprüft, ob es Einträge in der Datenbank gibt, die aufgrund des Retentionsintervalls gelöscht werden sollen.

Listing 5.6: `loginidsd` Hauptschleife

```
1  while(1) {
2      logMsg(9, "Waiting for input...");
3      sleep($config{loginidsdCheckInt});
4
5      logMsg(9, "Connecting to database.");
6      $dbh = DBI->connect($config{db_ds}, $config{db_user}, $config{db_password}, {
           RaiseError => 0, AutoCommit => 0 })
7              or logMsg(1, "Fatal error: Could not connect to database: " .
                   $DBI::errstr) and safeExit(1);
8
9      my $querySth = $dbh->prepare("SELECT
           id,service,timestamp,user,source,destination,success FROM LogEntries WHERE
           processed=0 ORDER BY timestamp");
10
11     my $rv = $querySth->execute;
12     if (!$rv) {
13         logMsg(3, "Database SELECT error: " . $DBI::errstr);
14         next;
15     }
16     my $entryCount=0;
17     while ( my ($id, $service, $timestamp, $user, $source, $destination, $success) =
           $querySth->fetchrow_array() ) {
18         logMsg(9, "Fetched: $id, $service, $timestamp, $user, $source, $destination,
               $success");
19         process_input($service, $timestamp, $user, $source, $destination, $success)
               or last;
20         dbDoTimeout("UPDATE LogEntries SET processed=1 WHERE id='" . $id . "'") or
               last;
21         $dbh->commit(); #everything done, commit changes after every processed entry
22         $entryCount++;
23     }
24
25     $querySth->finish();
26     undef $querySth;
27
28     logMsg(6, "Processed ".$entryCount." events") if($entryCount);
29
```

```
30      removeOldDbLogEntries();
31      removeOldUsersAndConnections();
32
33      $dbh->disconnect();
34      undef $dbh;
35      logMsg(9, "Database connection closed.");
36  }
```

Die Funktion `process_input()` übernimmt dann die Analyse des ihr übergebenen Log-Eintrags. Einzelheiten hierzu können direkt im Sourcecode nachgesehen werden. Die Überprüfung auf bestimmte Alarme ergibt sich durch Selektion der relevanten Datenbankfelder und Auswertung der Ergebnisse in if-Abfragen. Interessante Codeausschnitte zu speziellen Problemen und Anforderungen sind in anderen Abschnitten dieses Kapitels zu finden. Zum Beispiel ist in Kapitel 5.5 beschrieben, wie das Mapping von IP-Adressen auf bestimmte Zeichenketten funktioniert.

Eine Funktion, die nicht an anderer Stelle beschrieben wird, ist die `dbDoTimeout`-Funktion. Diese Funktion wurde entwickelt, um vor allem beim Zugriff auf eine `SQLite`-Datenbank fehlertoleranter zu sein. `SQLite` kennt nur Locks auf Datenbankebene, wodurch es häufig zu Programmabstürzen wegen zu langer Wartezeiten gekommen ist. Die Funktion probiert das Zugreifen auf die Datenbank nach einem „exponential backoff“-Verfahren. Der Rückgabewert dieser Funktion gibt an wie viele Tupel betroffen waren. Interessant dabei ist, dass die Funktion `0` im Fehlerfall und `0E0` bei Erfolg aber keinen betroffenen Tupel zurück gibt. Das hängt damit zusammen, dass Perl `0` als `false` interpretiert. In Exponentialschreibweise `0E0` hat der Rückgabewert den gleichen numerischen Wert, wird aber von Perl als `true` betrachtet.

Listing 5.7: dbDoTimeout-Funktion

```
1   sub dbDoTimeout($) {
2       my $query=shift;
3       my $timeout=5;
4       my $tryCount = 0;
5       my $affectedRows;
6       while(! ($affectedRows = $dbh->do($query))) {
7           logMsg(4, "Could not do $query: " . $DBI::errstr);
8           if($tryCount < 5) {
9               logMsg(6, "Retrying in $timeout seconds");
10              sleep($timeout);
11              $timeout = $timeout * 2;
12              $tryCount++;
13          } else {
14              logMsg(2, "DB->Do failed to often! Aborting");
15              $dbh->rollback();
16              return 0;
17          }
18      }
19      logMsg(9,"DB->Do($query) done.");
20      return $affectedRows if ($affectedRows >= 0); # Depending on database driver in
            some cases this value could also be -1
21      return 1;
22  }
```

5.8. Retentionsintervall

Die verschiedenen Retentionsintervalle, die in Kapitel 4.5 beschrieben wurden, werden in verschiedenen Programmteilen umgesetzt. Die Methoden zum Löschen von nicht mehr gebrauchten Logdateien werden innerhalb von `logindexd` realisiert, da dieses Programm auch für das Einlesen der Logdateien zuständig ist. Das Löschen von Datenbankeinträgen hingegen wird von `loginidsd` durchgeführt, weil `logindexd` selber keine Verbindung zur Datenbank herstellt. Das Löschen alter Logdateien wird in Kapitel 5.8.1, das alter Datenbankeinträge in Kapitel 5.8.2 gezeigt.

5.8.1. Löschen alter Logdateien

Das Löschen alter Logdateien wird von `logindexd` in der Funktion `removeOldFiles()` durchgeführt. Diese Funktion wird immer nach der Verarbeitung von neuen Logdateien ausgeführt und löscht die Logdateien, die in dem `logfiles/done` und `logfiles/parts` Ordner liegen. In diese Ordner werden die Logdateien nach einer erfolgreichen Bearbeitung verschoben. Das Datum, das verwendet wird um das Alter eines Logfiles zu bestimmen, ist durch den im Dateinamen verwendete Zeitstempel gegeben. Das hier verwendete Retentionsintervall ist `retInt`.

Listing 5.8: Algorithmus zum Löschen alter Logdateien

```
1  sub removeOldFiles() {
2      my $retInt = 7; # default seven day interval
3      $retInt = $config{retInt} if(defined($config{retInt}));
4      return if($retInt < 0); # delete nothing if retInt below zero
5      my @doneFiles = (<$doneDir/*>); # get files from the done directory
6      my @partsFiles = (<$partDir/*>); # get files from parts directory
7      push(@doneFiles, @partsFiles); # merge done and parts files
8      my @sortedFiles = sort{&compareFileTimestamps}@doneFiles; # sort the files
9      my $nowDate = DateTime->now; # get current time
10     my $retDate = DateTime->now->subtract(days => $retInt);
11     logMsg(9, "removeOldFiles: now is: $nowDate retDate is: $retDate");
12
13     foreach my $file (@sortedFiles){
14         my $fileDate;
15         if ($file =~ /[^-]*[-][^-]*[-]([\dT:-]*)(\..*)?$/) {
16             $fileDate = DateTime::Format::Flexible->parse_datetime($1);
17         } else {
18             logMsg(3,"Could not determine timestamp: ignoring $file");
19             next;
20         }
21         if(DateTime->compare($retDate, $fileDate) == 1){
22             # file date is before delete date
23             logMsg(5, "Deleting old file: $file");
24             unlink($file) or logMsg(3, "Could not delete file: $file. " . $!);
25         } else {
26             logMsg(5, "Not deleting file: $file and newer files");
27              # list of files is reverse sorted by date
28             last; # if we found the first file not to delete were finished
29         }
30     }
31 }
```

5.8.2. Löschen alter Datenbankeinträge

Genauso wie alte Logdateien sollen die Einträge in der Datenbank nicht unbegrenzt gespeichert werden. Bei den Datenbankeinträgen gibt es aber mehrere Abstufungen, wann etwas gelöscht wird. Nach dem Retentionsintervall `retInt` werden alle Einträge aus `LogEntries`, das die aus den Logdateien extrahierten Logzeilen enthält, und die Einträge des `DetailedLog`, das eine detaillierte, nicht aggregierte Version der Logeinträge für die manuelle Analyse enthält, gelöscht. Nach dem `forgetUserInt` werden Benutzer aus der Datenbank entfernt, die schon länger, als in dem Intervall angegeben, nicht mehr gesehen wurden. Genau so werden nach dem `forgetConInt` Einträge aus dem `LongtimeLog` entfernt. Dies wird von der Funktion, die in Listing 5.9 dargestellt ist, durchgeführt. Da die Syntax zur Bestimmung des Zeitintervalls in `SQLite` und `MySQL` unterschiedlich ist, müssen alle Löschbefehle doppelt implementiert werden und zur Laufzeit je nach verwendeter Datenbank der passende ausgewählt werden.

Listing 5.9: Algorithmus zum Löschen alter `LogEntries` und `DetailedLog` Einträge

```
1  sub removeOldDbLogEntries() {
2      my $retInt = 7; # default seven day interval
3      # if defined use value from config
4      $retInt = $config{retInt} if(defined($config{retInt}));
5      return if($retInt < 0); # delete nothing if retInt below zero
6      my $deletedLE = 0;
7      my $deletedDTL = 0;
8      $deletedLE = dbDoTimeout("DELETE FROM LogEntries WHERE processed=1 AND timestamp
           < DATE_SUB(NOW(), INTERVAL $retInt DAY);") or return 0 if ($config{db_ds} =~
           /^dbi:mysql:/);
9      $deletedLE = dbDoTimeout("DELETE FROM LogEntries WHERE processed=1 AND
           DATE(timestamp,'+$retInt day') < DATETIME('now');") or return 0 if
           ($config{db_ds} =~ /^dbi:SQLite:/);
10     $deletedDTL = dbDoTimeout("DELETE FROM DetailedLog WHERE timestamp <
           DATE_SUB(NOW(), INTERVAL $retInt DAY);") or return 0 if ($config{db_ds} =~
           /^dbi:mysql:/);
11     $deletedDTL = dbDoTimeout("DELETE FROM DetailedLog WHERE DATE(timestamp,
           '+$retInt day') < DATETIME('now');") or return 0 if ($config{db_ds} =~
           /^dbi:SQLite:/);
12     $dbh->commit();
13     logMsg(5,"Removed $deletedLE old entries from LogEntries and $deletedDTL from
           DetailedLog") if ($deletedLE > 0 or $deletedDTL > 0);
14 }
```

Das Löschen der Benutzer und Verbindungseinträge funktioniert nach dem selben Schema. Wenn ein Benutzer gelöscht wird, müssen aber auch alle ihm zugeordneten Verbindungen, Alarme und Verbindungsdetails gelöscht werden. Ebenso müssen beim Löschen einer Verbindung die zugehörigen Alarme und Verbindungsdetails gelöscht werden. Die Abhängigkeiten sind in der Übersicht zum zentralen Datenbankschema in Abbildung 4.3 zu sehen. Da es kein einheitliches Intervall für alle Benutzer gibt, werden die Benutzergruppen der Reihe nach durchgearbeitet. Dabei wird jeweils für die Benutzer der entsprechenden Gruppe das geltende `forgetUserInt` und `forgetConInt` berechnet und die Einträge, die älter als erlaubt sind, gelöscht.

5.9. Entwicklung eines Konvertierungsskriptes für Apache Webserver mod_auth

In Kapitel 4.1.2 wurde beschrieben, wie die Logdateien von SSH-Server und Apache Webserver mit `mod_auth` aussehen und wie diese prinzipiell zu verarbeiten sind. In diesem Kapitel wird nun gezeigt, wie diese Konvertierungsskripte aufgebaut sind.

Zu Anfang wird die Datenbankverbindung hergestellt und dazu über das `LoginIDS`.pm Modul die `LoginIDS`-Konfigurationsdatei gelesen. Dieser Abschnitt ist in Listing 5.10 dargestellt. Dieser Konfigurationsteil kann für alle Konvertierungsskripte identisch sein. Die Behandlung der Logeinträge wird erst in der Funktion `readLoop()` durchgeführt. Dies ist in Listing 5.11 dargestellt.

Listing 5.10: Konverterskript Setup-Teil

```
setupLogfile();
%config = readConfig();

$dbh = DBI->connect($config{db_ds}, $config{db_user}, $config{db_password},
                    { RaiseError => 0, AutoCommit => 1 })
            or logMsg(1, "Could not connect to database: " . $DBI::errstr)
            and safeExit(1);

open(I, "< " . $inputFile)
or logMsg(1, "Could not open input file " . $inputFile .": " . $!)
and safeExit(1);

$sth = $dbh->prepare("INSERT INTO LogEntries(service,timestamp,user,source,".
                     "destination,success) VALUES (?,?,?,?,?,?)");

readLoop();

close(I);
$sth->finish();
$dbh->disconnect();
$dbh=undef;
safeExit(0);
```

Listing 5.11: readLoop()-Funktion des Konverterskript für Apache access.log

```
while (my $line = <I>) {
    $line =~ s/[\r\n]//g; # chomp

    my ($timestamp, $host, $user, $source, $success);
    if ($line =~
        /^(.*)\s(.*)\s-\s(\w+)\s\[(\d{2}\/\w{3}\/\d{4}:\d{2}:\d{2}:\d{2}\s.\d{4})\]/)
        {
        $host = $1;
        $source = $2;
        $user = $3;
        $timestamp = $4;
        $success = 1;
    }

    next unless ((defined $user) && (length $user));

    $timestamp = normalizeTimestamp($timestamp);
    my $outputCopy = $output;
```

```
17      $outputCopy =~ s/TIMESTAMP/$timestamp/g;
18      $outputCopy =~ s/DESTINATION/$host/g;
19      $outputCopy =~ s/USER/$user/g;
20      $outputCopy =~ s/SOURCE/$source/g;
21      $outputCopy =~ s/SUCCESS/$success/g;
22
23      my $timeout=10;
24      my $tryCount = 0;
25      while(! $sth->execute( split(/,/, $outputCopy))) {
26          logMsg(3, "Could not execute: " . $DBI::errstr);
27          if($tryCount < 7) {
28              logMsg(4,print "Retrying in " . $timeout . "seconds");
29              sleep($timeout);
30              $timeout = $timeout * 2;
31          } else {
32              logMsg(1, "Execute failed to often! Aborting\nCurrent Line was: $line");
33              safeExit(1);
34          }
35      }
36      logMsg(7, $outputCopy);
37  }
```

5.10. Ausführen externer Skripte

In Kapitel 4.9 wurde beschrieben, wie der Aufruf von externen Skripten prinzipiell aussehen kann. Dieser Ablauf wurde in dem Sequenzdiagramm 4.6 schematisch dargestellt. In diesem Kapitel wird nun zuerst die Umsetzung innerhalb von `loginidsd` beschrieben und danach gezeigt wie ein `triggers.pl`-Skript aussehen kann.

Der Codeausschnitt in Listing 5.12 zeigt, wie das `triggers.pl`-Skript aufgerufen wird. Zu Beginn wird überprüft, ob das Skript ausführbar ist. Zum Deaktivieren der externen Skripte reicht es daher aus das `execute`-Bit zu entfernen. Wenn das Skript ausgeführt werden kann, wird es mit dem Alarmtyp und der Alarmmeldung als Parametern aufgerufen. Die erste Zeile des Standardoutput wird von `loginidsd` eingelesen. Über diese Zeile können zum einen Fehlermeldungen übergeben werden, die in das Logfile von `loginidsd` geschrieben werden, zum anderen kann, wie in Kapitel 4.11 beschrieben, bei einem `NEW USER` Alarm der Benutzertyp gesetzt werden. Um den Benutzertyp auf `1` zu setzten, muss das Trigger-Skript nur „`usertype = 1`“ in den Standardoutput schreiben. `loginidsd` verarbeitet den Standardoutput und setzt bei einer gültigen Antwort den Benutzertyp. Wie dieses Skript im Falle des vom LRZ verwendeten Systems zur Benutzerverwaltung aussieht, wird in Kapitel 5.10.1 gezeigt. Der von `loginidsd` gelesene Teil des Standardoutput ist auf eine Zeile beschränkt, um nicht zu viel Zeit mit dem Lesen und Loggen von externen Skripten zu brauchen.

Listing 5.12: Ausführen externer Skripte aus `loginidsd`

```
1  if( -x "./triggers/triggers.pl" ) {
2    open(TRIGGER, "perl ./triggers/triggers.pl '".$alarm_type."' '".$msg."' |")
3         or logMsg(3 , "Could not execute trigger: ".$!) and return 0;
4    my $triggerRet = <TRIGGER>;
5    close(TRIGGER);
6    if($alarm_type eq "NEW_USER" and defined($triggerRet)
7       and $triggerRet =~ /userType\s?=\s?(\d+)/)
8    {
```

```
9       my $userType = $1;
10      logMsg(8, "New User Trigger set usertype of ".$username." to ".$userType);
11      dbDoTimeout("UPDATE Users SET type='".$userType."' WHERE id='".$userId."';")
12          or return 0;
13    }
14    logMsg(6, "Trigger returned: $triggerRet") if defined($triggerRet);
15  }
```

Das `triggers.pl`-Skript liest dann den Alarmtyp und die Alarmmeldung ein und ersetzt sein Prozessimage durch das in der switch-Anweisung ausgewählte Skript. Die Verwendung von `exec` vereinfacht das Übergeben des Standartoutputs an `loginidsd`. Nachdem `triggers.pl` durch das spezielle Skript ersetzt wurde, geht dessen Standardoutput direkt an `loginidsd`. Bei einer Implementierung mit der `system`- oder `open`-Funktion müsste das `triggers.pl`-Skript nach der Ausführung des speziellen Skriptes dessen Standardoutput speichern und vor dem Beenden selber in den Standardoutput schreiben.

Die beiden speziellen Skripte, die in dieser Arbeit implementiert wurden, werden in den nächsten zwei Unterkapiteln 5.10.1 und 5.10.2 beschrieben.

Listing 5.13: Verarbeitung von Alarmen in triggers.pl

```
1  my $alertType = shift;
2  my $alertMessage = shift;
3
4  switch($alertType){
5      # If the alert type is NEW USER replace this process with the trigger
6      # that handles NEW USER alerts and returns a userType uncomment to enable
7      case("NEW_USER") { exec("./triggers/newUser.pl '".$alertMessage."'")
8              or die "Could not execute NEW_USER trigger script: " .$!; }
9      case("EXPIRED_USER") { exec("./triggers/expiredUser.pl '".$alertMessage."'")
10             or die "Could not execute EXPIRED_USER trigger script: " .$!; }
11 }
12
13 exit(0);
```

5.10.1. Benutzervalidierung

Wie das Listing 5.13 zeigt, wird bei einem `NEW USER` Alarm das `newUser.pl`-Skript aufgerufen. Dort wird wie in Kapitel 4.11 beschrieben, der Benutzername in einer Liste von dem zentralen LDAP-Verzeichnis bekannten Benutzern gesucht. Die dort angegebenen Gruppenmitgliedschaften werden verwendet, um den Benutzer auf die in `LoginIDS` verwendeten Benutzertypen abzubilden. Wenn ein Benutzer nicht gefunden werden kann, handelt es sich entweder um eine Funktionskennung wie „root“ oder einen dem LDAP-System nicht bekannten Benutzer. Die beiden Fälle stellen eine Ausnahme dar, die wie auch in Kapitel 3.2.4 beschrieben, nicht sehr häufig vorkommen sollte. Daher wird für diese Fälle eine besondere Benutzergruppe vergeben, bei der das Retentionsintervall höher ist, es keine Lernphase gibt und jeder Login gemeldet wird.

5.10.2. Mail-Benachrichtigungen

Da der `EXPIRED USER` Alarm nur für die Trigger-Skripte gemeldet wird, sollte er dort auch behandelt werden. Im Kapitel 4.4.1 wurde erklärt, warum dieser Alarm nicht dauerhaft in der Datenbank gespeichert wird. Um einen Verantwortlichen darauf aufmerksam zu machen, dass es einen längere Zeit ungenutzten Benutzeraccount gibt, der je nach vorhandenen Richtlinien unter Umständen deaktiviert werden sollte, wird eine Mail-Benachrichtigung implementiert.

6. Visualisierung in Splunk

6.1. Funktionalität von Splunk

Splunk ist ein Log-Management-System, mit dem sich nahezu alle Arten von Logfiles aufbereiten lassen. Die wichtigsten Funktionen und Möglichkeiten, die der Einsatz von Splunk bietet, werden im folgenden Abschnitt vorgestellt.

Bedient wird Splunk über eine Webschnittstelle und kommt daher ohne Installation einer weiteren Clientsoftware aus. Über die Webschnittstelle bietet Splunk die Möglichkeit, die indizierten Daten mit diversen grafischen Elementen aufzubereiten. Dazu stehen Tabellen, Graphen und verschiedene Diagramme zur Verfügung. Neben der grafischen Aufbereitung der Logeinträge bietet Splunk auch eine mächtige Suchfunktion, mit der die indizierten Daten auf bestimmte Eigenschaften hin gefiltert werden können. Auch können diese Grafiken und Suchergebnisse in nahezu Echtzeit mit dem Eintreffen neuer Ereignisse aktualisiert werden.

Zum Testen gibt es von Splunk eine Demoversion, die nach Ablauf des Testzeitraums in eine freie Version umgewandelt werden kann. Der freien Version fehlen einige Funktionen, wie zum Beispiel die Benutzerverwaltung und es kann ein maximales Datenvolumen von 500MB pro Tag indiziert werden. Zum Entwickeln von LoginIDS hat diese Version aber ausgereicht. Die während der Arbeit verwendeten Splunk-Versionen waren 4.3.1 bis 4.3.3. Weitere Informationen und eine ausführliche Dokumentation findet sich unter http://www.splunk.com.

Im Folgenden werden einige Vor- und Nachteile, die der Einsatz von Splunk bietet, diskutiert. Der größte Nachteil ist, dass die von LoginIDS in einer SQL-Datenbank gesammelten Daten nicht direkt in Splunk verwendet werden können. Stattdessen müssen sie in die Datenbank von Splunk importiert werden, wodurch sehr ähnliche Daten an zwei Orten gespeichert werden. Der Import ist dabei über einen gescripteten Input möglich. Dieses Verfahren wird in Abschnitt 6.2.1 vorgestellt. Dadurch entsteht aber eine statische Kopie der LoginIDS-Datenbank, was nicht nur von der Übersichtlichkeit her unschön, sondern auch Probleme bereitet, wenn Einträge in der LoginIDS Datenbank geändert werden. Der Importvorgang sieht nur das einmalige Importieren eines Datenbankeintrages vor, wodurch nachträgliche Änderungen nicht übernommen werden. Die einzige Möglichkeit, dieses Problem zu umgehen, wäre es, nicht nur neue Einträge inkrementell zu importieren, sondern den gesamten Datenbestand in Splunk zu löschen und alles neu zu importieren. Eventuell wäre eine Lösung praktikabel, bei der einmal am Tag alles neu importiert wird und ansonsten nur inkrementelle Updates gemacht werden. Ideal ist das aber auch nicht, da so unter Umständen inkonsistente

Daten angezeigt werden. Als Beispiel sei hier der `lastSeen`-Zeitstempel, der für jeden Benutzer gespeichert wird, genannt. Dieser Zeitstempel würde nur bei einem kompletten Reimport des Datenbankbestandes aktualisiert werden. Neue Alarmmeldungen oder Loginmeldungen für den Benutzer würden unter Umständen auch schon bei einem normalen Update importiert. Dann wäre es verwirrend, wieso der Benutzer einen Login durchgeführt hat, aber, aus Sicht von `Splunk`, dabei nicht sein `lastSeen`-Zeitstempel aktualisiert wurde. Um diese Inkonsistenzen zu vermeiden, werden daher aktuell nur unveränderliche Felder in `Splunk` importiert.

Die oben genannte Schwäche im Zusammenspiel von `LoginIDS` und `Splunk` wird durch die sehr einfache und grafisch schöne Aufbereitung der importierten Daten ausgeglichen. Im Abschnitt 6.2.2 wird das User-Interface, dass durch die `Splunk` App realisiert wird, genauer beschrieben.

6.2. Entwurf der `LoginIDS-Splunk` App

Wie als Anforderung in Kapitel 3.2.2 beschrieben, ist die Visualisierung der Ausgabe von `LoginIDS` durch eine `Splunk` App ein wichtiger Punkt innerhalb dieser Arbeit. Ohne eine geeignete Möglichkeit zur Anzeige der Ergebnisse, ist die Analyse nur wenig nützlich.

6.2.1. Dateninput

Damit die Analysedaten von `LoginIDS` in `Splunk` angezeigt werden können, müssen sie zunächst importiert werden. Zum Importieren von Informationen bietet `Splunk` zwei verschiedene Möglichkeiten. In Abbildung 6.1 sind beide schematisch dargestellt. Die einfachste Methode ist der Import direkt aus einem Logfile. Diese Methode ist in der Abbildung 6.1 links dargestellt. Der Nachteil hierbei ist die Einführung eines weiteren Logfiletyps, der nur zur Datenübertragung an `Splunk` verwendet wird. Dann würden Logfiles aus den Anwendungen in ein Logfile, das von `LoginIDS` gelesen werden kann, übersetzt, nur um wieder als ein dritter Typ von Logfile von `Splunk` eingelesen zu werden. Zudem würde dadurch die Information über Alarmmeldungen inkonsistent zwischen der `LoginIDS` Datenbank und der `Splunk`-Datenbank verteilt. Auf der rechten Seite der Abbildung 6.1 ist der Ablauf dargestellt, bei dem die Alarmmeldungen über ein Skript direkt von `Splunk` indexiert werden.

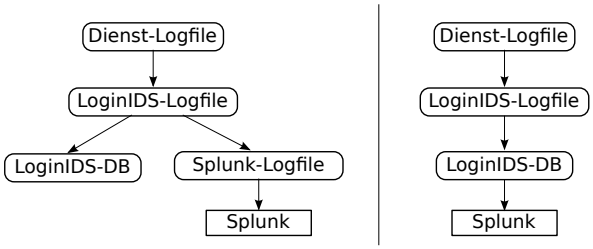

Abbildung 6.1.: Methoden zur Analyse von Logfiles und Import des Ergebnisses in `Splunk`

Das Ziel ist es daher, alle Informationen in der LoginIDS Datenbank zu speichern und über ein Skript von Splunk aus darauf zuzugreifen. Dazu ist in Kapitel 4.3 das Datenbankschema um eine Tabelle zur Speicherung aller Alarmmeldungen erweitert worden.

Um die gewünschten Informationen aus der Datenbank lesen zu können, wurde ein Perl-Skript geschrieben, das es ermöglicht, neue Einträge aus der Datenbank zu importieren. Dieses Skript queryDatabase.pl merkt sich die ID des letzten indexierten Datenbankeintrags und gibt nur neuere, dass heißt, Einträge mit einer höheren ID als Komma-Separated-List aus. Dieses Skript befindet sich, wie alle von der Splunk App ausführbaren Programme, im Ordner $SPLUNK_HOME/etc/apps/loginIDS/bin/.

Beim Start des Skriptes werden die Datenbankeinstellungen aus der Konfigurationsdatei $SPLUNK_HOME/etc/apps/loginIDS/local/loginIDS.conf gelesen. Diese Datei wird beim Setup-Prozess in Splunk erstellt. Bei einem Aufruf des Skriptes müssen zwei Parameter, der Pfad zu der Datei, in der die letzte Position $lastPosition vermerkt ist und die Datenbankanfrage $query, übergeben werden. Ein Beispiel für so einen Query ist in der letzten Zeile von Listing 6.2 zu sehen. Das Keyword <lastPostion> wird dabei im Skript durch die ID des aus der Datei, in der die Position des letzen gelesenen Eintrages steht, ersetzt.

In Listing 6.1 ist die zentrale Funktion dieses Skriptes gezeigt. Zuerst wird die SQLite Datenbank geöffnet. In der zweiten Zeile ist das Ersetzen des Keywords <lastPosition> zu sehen. Danach wird der Query ausgeführt und in der for-Schleife durch alle Ergebnisse iteriert. Die innere for-Schleife ersetzt nicht definierte Inhalte, also Felder, die in der Datenbank den Wert NULL hatten, durch einen leeren String. Die darauf folgende print-Anweisung gibt das aktuelle Tupel auf den Standardoutput aus. Innerhalb der Konfiguration des Skriptes wurde die besondere Perl-Variable $", die angibt, durch welchen Separator Elemente eines Arrays bei einem print-Aufruf ausgegeben werden, auf "," gesetzt. Dadurch werden die selektierten Felder als Komma-Separated-List ausgegeben. Zum Schluss wird noch in der Variable $newPosition die ID des neuen, zuletzt gelesenen Elements gespeichert und die Datenbank geschlossen. Wie im Code zu sehen ist, muss die ID immer das erste selektierte Feld sein! Das schränkt die allgemeine Anwendbarkeit des Programmes aber nicht ein. Vielmehr kann durch Umstellen des SELECT-Ausdrucks prinzipiell jedes geeignete Feld, unabhängig von seinem Namen, als „ID-Feld“ verwendet werden. Nachdem alle Elemente verarbeitet wurden, wird die $newPosition in die Datei, in der die letzte verarbeitete Position gespeichert wird, geschrieben.

Listing 6.1: Verarbeitung der Datenbankanfrage und Ausgabe der Ergebnisse

```
1  my $lastPosition = 0;
2  if (open(LAST, "< ".$lastRead)) {
3      $lastPosition = <LAST>;
4      chomp($lastPosition);
5      close(LAST);
6  }
7
8  my $newPosition = $lastPosition;
9  my $dbh = DBI->connect($DB_data_source, $DB_username, $DB_password, { RaiseError =>
       1, AutoCommit => 1 }) or die "Could not connect to database: " . $DBI::errstr .
       "\n";
10 my ($id, $timestamp, $falsePositive, $alertName, $service, $source, $destination,
       $loginName);
```

```
11 $query =~ s/<lastPosition>/$lastPosition/;
12 my $res = $dbh->selectall_arrayref($query) or die "Could not fetch data from
       database: " .$DBI::errstr . "\n";
13 foreach my $row (@$res){
14     foreach (@$row) { $_ = '' unless defined }; # Define values that where NULL in
           the database
15     print "@$row\n";
16     $newPosition=@$row[0];
17 }
18 $dbh->disconnect();
19
20 open(NEW, ">".$lastRead) or die "# Could not write to ".$lastRead."\n";
21 print NEW $newPosition."\n";
22 close(NEW);
```

Dadurch, dass dieses Programm sehr flexibel in Bezug auf die Anfrage, die Anzahl der selektierten Felder und der Datenbank ist, lassen sich damit alle nötigen Informationen indexieren. Dazu sind kurze Wrapper-Skripte geschrieben worden, die von `Splunk` regelmäßig aufgerufen werden. Diese Skripte befinden sich auch im `bin`-Ordner der `Splunk` App. Ein Beispiel für so ein Skript ist in Listing 6.2 zu sehen.

Listing 6.2: Aufruf von `queryDatabase.pl` in `getLongtimeLogs.sh`

```
1 /opt/splunk/etc/apps/loginIDS/bin/queryDatabase.pl \
2 "/opt/splunk/etc/apps/loginIDS/bin/.lastDetailedLogRead" \
3 "SELECT id,timestamp,service,userId FROM DetailedLog WHERE id > <lastPosition>;"
```

Die Verarbeitung der Ausgabe des `queryDatabase.pl` Programms in `Splunk` wird durch die Konfigurationsdateien im Ordner `$SPLUNK HOME/etc/apps/loginIDS/default/` spezifiziert. Die Konfiguration des Dateninput wird in den Dateien `inputs.conf`, `props.conf` und `transforms.conf` vorgenommen. Im Folgenden wird kurz gezeigt, wie die einzelnen Konfigurationen für das Einlesen des `LongtimeLog`s funktionieren. Die anderen Fälle sind analog realisiert.

Eine neue Datenquelle muss zuerst in der `inputs.conf` definiert werden. In Listing 6.3 ist dafür ein Beispiel zu sehen. Die erste Zeile mit den eckigen Klammern zeigt den Beginn einer neuen Input-Definition an. Zusätzlich ist zu erkennen, dass der Typ des Inputs ein Skript ist, das sich im `bin`-Ordner der `LoginIDS`-App befindet. Die zweite Zeile gibt das Intervall an, in dem das Skript aufgerufen werden soll. Hier ist das Intervall durch ein Cron-Schedule, der die Ausführung jeden Morgen um 5 Uhr plant, gesetzt. Alternativ kann auch ein Intervall in Sekunden angegeben werden. Danach wird noch der Sourcetype und der zu verwendende `Splunk`-Index angegeben. Diese Felder helfen die Daten von anderen Datenquellen zu trennen. `LoginIDS` verwendet für alle Daten den `Splunk`-Index „loginids“ aber unterschiedliche Sourcetypes.

Listing 6.3: Beispiel eines Eintrages in `inputs.conf`

```
1 [script:///opt/splunk/etc/apps/loginIDS/bin/getLongtimeLogs.sh]
2 interval = 0 5 * * *
3 sourcetype = loginIDS_LongtimeLog
4 index = loginids
```

In der `inputs.conf` wurden die Einstellungen für den Input getroffen. Die Optionen, die einen Sourcetype festlegen, werden in der Datei `props.conf` angegeben. Listing 6.4 zeigt hier

weiter das Beispiel des LongtimeLogs. Die eckigen Klammern zeigen auch hier den Beginn der Optionen für einen bestimmten Sourcetype an. Die zweite Zeile setzt SHOULD_LINEMERGE auf false. Damit wird Splunk jede Zeile, die eingelesen wurde, einzeln betrachten. Das ist vor allem dann wichtig, wenn kein Zeitstempel in den Daten vorkommt. Splunk sucht dann in der Standardeinstellung über mehrere Zeilen nach einem Zeitstempel. In diesem Fall gibt queryDatabase.pl aber in jeder Zeile einen Eintrag aus der Datenbank aus. Die letzte Zeile verweist darauf, wie die einzelnen Felder aus der eingelesenen Zeichenkette extrahiert werden sollen.

Listing 6.4: Beispiel eines Eintrages in props.conf

```
1 [loginIDS_LongtimeLog]
2 SHOULD_LINEMERGE=false
3 REPORT-loginIDSLontimeLogExtractions = loginIDS_LongtimeLog_extractions
```

Wie die Extraktion der einzelnen Felder genau aussieht, wird in der Datei transforms.conf angegeben. Auch hier ist in Listing 6.5 wieder ein Beispiel zum LongtimeLog gezeigt. Mit dem Wert von DELIMS wird angegeben, dass die Eingabe komma-separated ist. Die FIELDS geben an, wie die einzelnen Felder im Splunk-Frontend identifiziert werden sollen. Die Liste deckt sich mit den in der SQL-Anfrage in Listing 6.2 selektierten Feldern.

Listing 6.5: Beispiel eines Eintrages in transforms.conf

```
1 [loginIDS_LongtimeLog_extractions]
2 DELIMS = ","
3 FIELDS = "Longtime-Id", "User-Id", "Service", "Source", "Destination"
```

Falls aus dem Splunk-Frontend Änderungen an den Einstellungen der oben genannten Konfigurationsdateien gemacht werden, speichert Splunk diese in dem Ordner $SPLUNK HOME/etc/apps/loginIDS/local/. Die dort festgelegte Einstellungen „überschreiben“ die in dem default-Ordner gemachten Angaben.

6.2.2. Visualisierung durch Dashboards und Forms

Ein wichtiger Punkt, der in der Anforderung in Abschnitt 3.2.2 beschrieben wurde, ist die Darstellung der Analyseergebnisse von LoginIDS. Die dazu erstellte Übersichts- und Startseite der App ist in Abbildung 6.2 zu sehen. Hier sollen demjenigen, der die Analyse auswertet, zu Beginn gleich wichtige Indikatoren für Fehler zur Verfügung stehen. Zusätzlich kann ausgewählt werden, aus welchem Zeitraum Alarmmeldungen angezeigt werden sollen. Gleich in der ersten Zeile wird gezeigt, wie viele Alarmmeldungen und Logevents innerhalb des ausgewählten Zeitraums verarbeitet wurden. Sollte bei der Verarbeitung von Logfiles ein Fehler auftreten und keine Logeinträge mehr analysiert werden, würde dies direkt auffallen. Darunter ist die Liste der festgestellten Alarme zu sehen. Hier ist die Liste etwas gekürzt und einzelne Felder sind anonymisiert worden. In der echten Anwendung ist die Liste 25 Zeilen lang und alle Felder sind sichtbar. Die letzte Grafik zeigt den Verlauf der Alarmmeldungen der letzten sieben Tage aufgespalten nach dem Alarmtyp. In der untersten Zeile befinden sich

drei Tortendiagramme, die die Verteilung der Alarme auf die Dienste, Ziel- und Quelladressen zeigen. Wenn das Quelladressen-Diagramm in einem eng begrenzter Zeitraum betrachtet wird, ist dessen Aussagekraft höher.

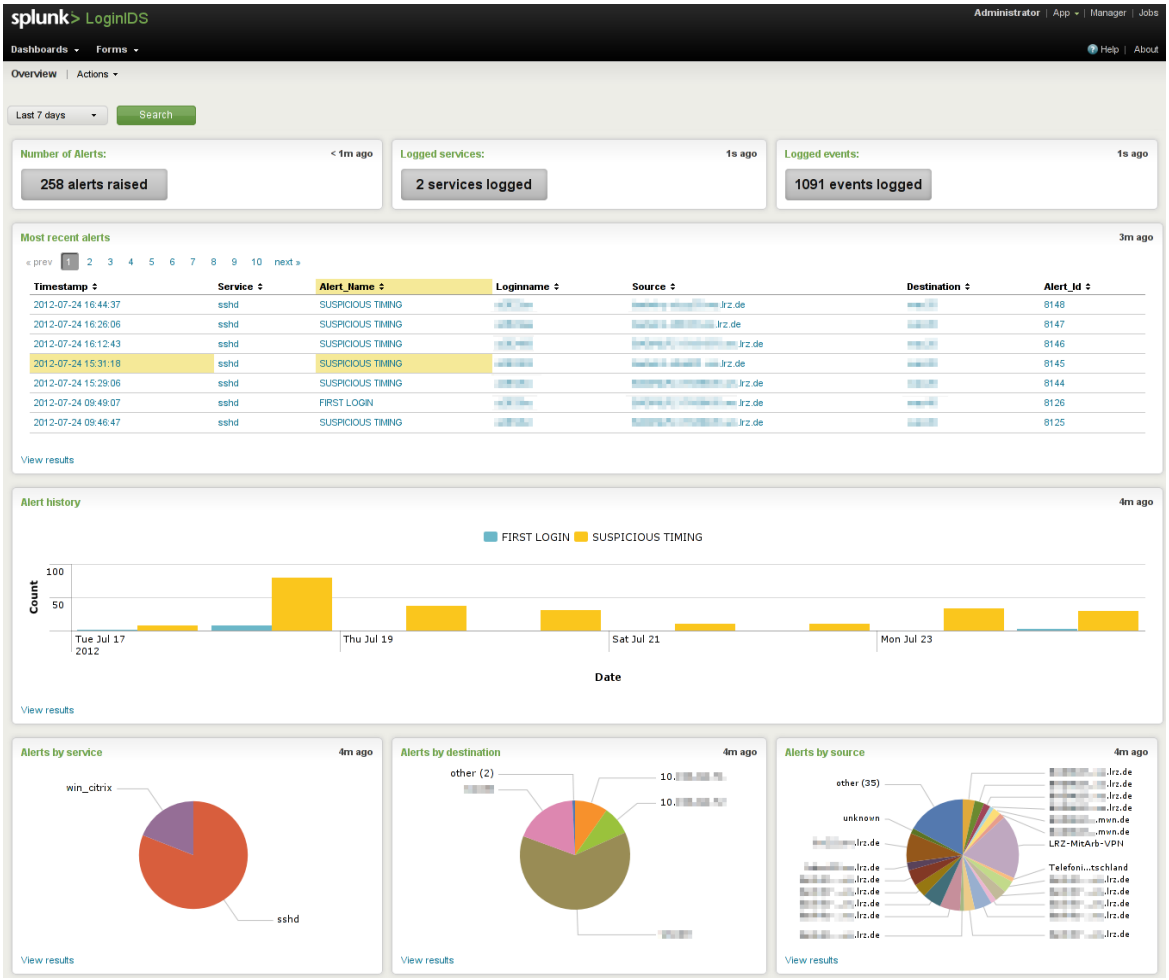

Abbildung 6.2.: Startdarstellung der `LoginIDS` `Splunk` App

Detailliertere Informationen über die Effizienz von `LoginIDS` gibt die Seite „Efficiency“, die in Abbildung 6.3 dargestellt ist. Die linke Grafik zeigt das Verhältnis zwischen Alarmmeldungen und verarbeiteten Logeinträgen. Die rechte Grafik zeigt wie viele Logeinträge für die verschiedenen Dienste eingegangen sind. Diese Seite bietet außerdem die Möglichkeit über das Drop-Down-Menü am oberen Rand nach speziellen Diensten zu filtern und die zu betrachtete Zeitspanne einzustellen.

Um den Alarm eines Benutzers genauer untersuchen zu können, gibt es eine Suche, die zu einem Benutzernamen alle Alarmmeldungen und Verbindungen heraussucht. Dieser Dialog ist in Abbildung 6.4 zu sehen. Die Grafik rechts unten gibt einen Überblick über das Loginverhalten des Benutzers, indem sie zeigt, an welchen Tagen und zu welcher Zeit Logins verzeichnet wurden. Die Tage, als 1–7 kodiert, entsprechen dabei Montag – Sonntag.

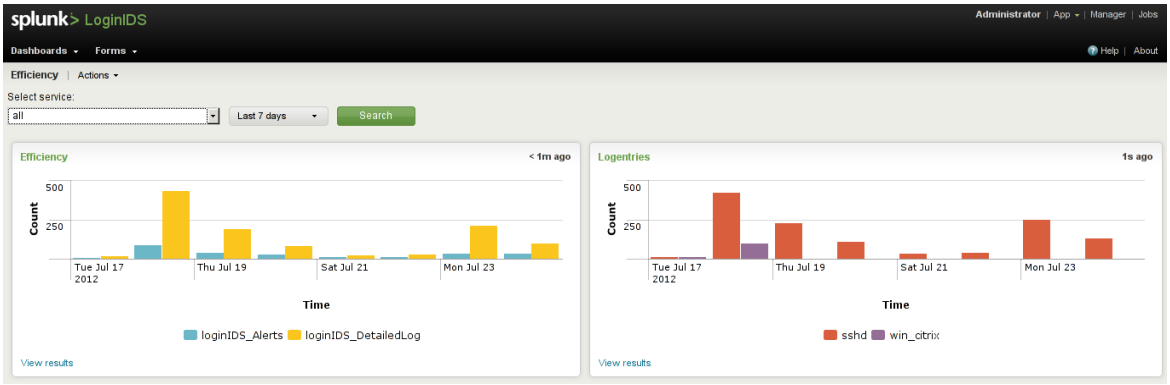

Abbildung 6.3.: Efficiency-Dashboard der `LoginIDS` `Splunk` App

Die Informationen, die in der `LoginIDS` `Splunk` App angezeigt werden, sind so aufbereitet, dass sie übersichtlich und einfach zu analysieren sind. Um alle von `LoginIDS` gesammelten Informationen sehen zu können, muss aber direkt auf der `LoginIDS`-Datenbank gearbeitet werden.

6.3. `Splunk`-Deployment-Optionen

Bei der Einrichtung von `LoginIDS` mit `Splunk`, gibt es prinzipiell drei mögliche Setups. Die einfachste Möglichkeit ist es, sowohl `LoginIDS` als auch `Splunk` auf einem System zu betreiben. Diese Methode wird in dem Unterkapitel 6.3.1 beschrieben. Die zweite Möglichkeit wäre es `LoginIDS` und `Splunk` auf unterschiedlichen System zu installieren und über eine `MySQL`-Datenbank die Informationen auszutauschen. Dies wird in dem Unterkapitel 6.3.2 gezeigt. Die dritte Alternative ist eine Mischung aus beidem, bei der `LoginIDS` und `Splunk` prinzipiell auch auf verschiedenen Hosts laufen. Auf dem `LoginIDS`-Host wird aber ein `Splunk`-Forwarder installiert, der die Weitergabe der Ereignisse an die Haupt-`Splunk`-Instanz übernimmt. Diese Möglichkeit wird in Unterkapitel 6.3.3 genauer erläutert. Diese Möglichkeiten werden in dem Kapitel 6.3.4 miteinander verglichen.

6.3.1. `LoginIDS` und `Splunk` auf dem selben Host

Diese Methode ist am besten geeignet, wenn `LoginIDS` zu Testzwecken ausprobiert werden soll und noch keine `Splunk`-Installation besteht, die verwendet werden kann. In dieser Konfiguration ist es möglich sowohl `SQLite` als auch `MySQL` als Datenbank einzusetzen. Die Installation ist hierbei einfach, da es ausreichend ist, `LoginIDS` an einer beliebigen Stelle zu entpacken und über das Setup-Skript zu konfigurieren. Danach kann `Splunk` und die `Splunk` App installiert werden, die ebenfalls leicht über einen Konfigurationsdialog eingerichtet werden kann. Zur Installation der `Splunk` App gibt es in Anhang C eine Anleitung. Ein Vorteil dieser Installation ist außerdem, dass keine Daten über das Netzwerk übertragen werden

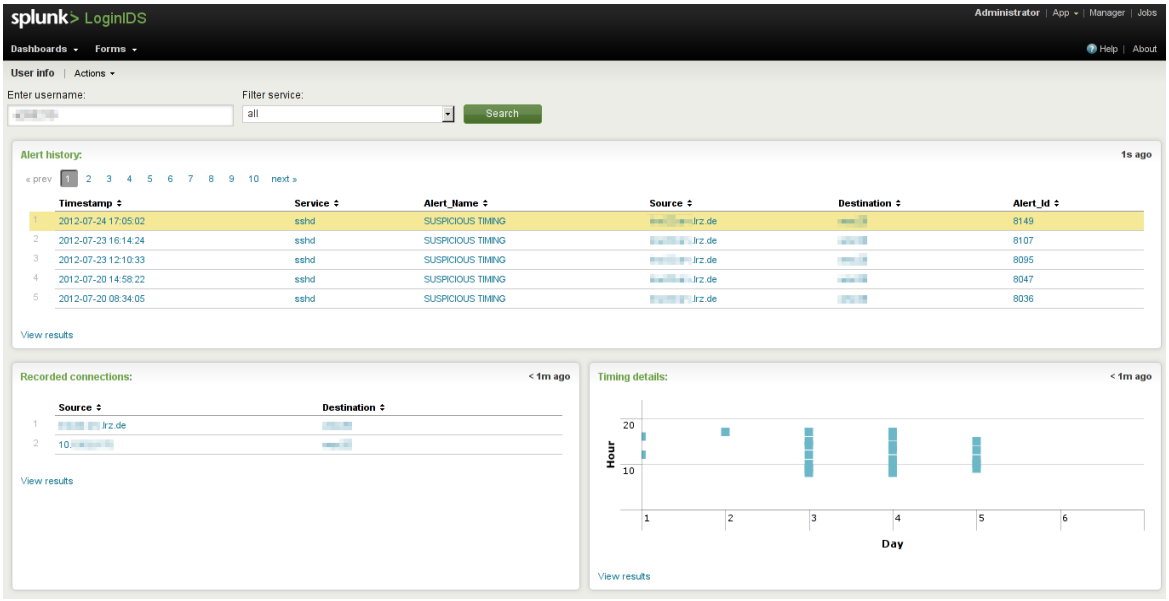

Abbildung 6.4.: Detailinformationen zu einem Benutzer in der `LoginIDS` `Splunk` App

müssen. Die Möglichkeit, mit `MySQL` oder einem `Splunk`-Forwarder Daten auf einen entfernten Host zu übertragen, sind ohne weitere Konfiguration und Änderungen an `LoginIDS` nicht verschlüsselt oder authentifiziert und daher potentiell anfällig für Manipulationen.

6.3.2. `LoginIDS` und `Splunk` auf eigenem Hosts ohne `Splunk`-Forwarder

Diese Konfiguration von `LoginIDS` und `Splunk` eignet sich für eine Umgebung in der schon ein `Splunk`-Server besteht, auf dem die Ergebnisse angezeigt werden sollen. Ohne einen `Splunk`-Forwarder bleibt hier allerdings nur die Möglichkeit eine netzwerkfähige Datenbank zu verwenden. Die einzig dazu von `LoginIDS` unterstützte Datenbank ist `MySQL`. Hier kann die `Splunk` App, wie in Anhang C beschrieben, installiert werden. Auch an der Installation von `LoginIDS` ändert sich nichts, das Datenbanksystem ist in diesem Fall für die Übertragung von Informationen über das Netzwerk verantwortlich. Da die Verbindung zur `MySQL`-Datenbank standardmäßig nicht verschlüsselt ist, eignet sich diese Methode nur, wenn die Daten über ein vertrauenswürdiges Netzwerk versendet werden.

6.3.3. `LoginIDS` und `Splunk` auf eigenem Hosts mit `Splunk`-Forwarder

Diese Konfiguration hat den höchsten Konfigurationsaufwand. Dies wird zusätzlich komplizierter, da es drei verschiedene Arten von `Splunk`-Forwardern gibt. Eine gute Übersicht über alle Optionen gibt das `Splunk` „Distributed Deployment Manual“[Spl12]. Die Forwarder-Versionen „Universal forwarder“ und „Light forwarder“ unterscheiden sich kaum. Wenn auf dem Host Python installiert ist, sollte der „Universal forwarder“ gewählt werden, da die

Splunk-Dokumentation den „Light forwarder“ als obsolet bezeichnet [Spl12, Seite 8]. Der dritte Typ ist der „Heavy forwarder“, der nahezu einer kompletten Splunk Installation entspricht, in der einige Funktionen deaktiviert wurden.

Der „Universal forwarder“ eignet sich, wenn die indizierten Daten nur weitergeleitet werden sollen. Da er kein Webinterface hat, muss die Konfiguration über die Kommandozeile erfolgen. Durch die geringere Funktionalität ist der „Universal forwarder“ kleiner und verbraucht weniger Ressourcen. Wird eine eigene Splunk-Instanz zum Weiterleiten verwendet, können dort auch die Ergebnisse angezeigt werden. Außerdem kann das Weiterleiten über das Webinterface konfiguriert werden. Detaileinstellungen müssen aber auch hier in den Konfigurationsdateien mit einem externen Editor geschehen.

6.3.4. Vergleich der Optionen

Wie schon in dem Kapitel 6.3.1 erwähnt, ist die Konfiguration von LoginIDS und Splunk auf ein und dem selben Host am einfachsten. Es müssen nur auf einem System die nötigen Perl-Module installiert werden. Als Datenbank kann sowohl SQLite als auch MySQL verwendet werden.

Wenn LoginIDS und Splunk nicht auf dem selben Host laufen sollen, ist die in Kapitel 6.3.3 beschriebene Methode mit einem der von Splunk angebotenen Forwarder unter Umständen der Methode aus Kapitel 6.3.2 vorzuziehen. Die Verwaltung und Installation von zwei Splunk-Instanzen muss dabei gegen das Installieren der für LoginIDS nötigen Perl-Module abgewogen werden. Die Perl-Module werden nur dann benötigt, wenn die Scripted-Inputs der LoginIDS-App aktiviert sind. Das heißt, dass bei einer Übertragung mittels Datenbank sowohl auf dem LoginIDS-Host als auch auf dem Splunk System die nötigen Module installiert werden müssen. Wenn ein Forwarder verwendet wird, müssen die Scripted-Inputs auf der entfernten Splunk-Instanz deaktiviert und die Perl-Module nicht notwendigerweise installiert sein. Außerdem kann das Forwarding von Splunk ohne Änderungen an LoginIDS so konfiguriert werden, dass es verschlüsselt stattfindet.

7. Test und Evaluation

In diesem Kapitel werden die Tests, die über die gesamte Dauer der Arbeit stattgefunden haben, beschrieben. Dabei wird auch darauf eingegangen, welche Veränderungen die besten Ergebnisse erzielt haben. In Kapitel 7.1 werden die beiden zum Testen genutzten Testumgebungen beschrieben.

7.1. Beschreibung der Testsysteme

Während der Arbeit wurde `LoginIDS` ständig betrieben und getestet. Dazu wurde zum einen ein am LRZ betriebenes System verwendet, auf dem mit realen Daten gearbeitet wurde. Zum anderen bestand eine lokale Testumgebung, um Funktionstests durchzuführen bevor Änderungen in das System am LRZ, das einem „Produktiv-System“ nahe kam, und das Git-Repository eingebracht wurden.

7.1.1. Testsystem am LRZ

Wie bereits in anderen Abschnitten der Arbeit ausgeführt, wurden in dem System am LRZ Daten von zwei `SSH`-Server und einem Citrix-Windows-Terminalserversystem ausgewertet. Das hierbei verwendete System ist eine virtuelle Maschine auf Basis von SUSE-Enterprise-Server.

Die Abbildung 7.1 gibt einen Überblick über die Anzahl der gemeldeten Alarme, wie sie in `Splunk` dargestellt werden. Zu erkennen ist, dass die Anzahl der `SUSPICIOUS LOGIN` Alarmmeldungen nicht zurück gegangen ist. Dies ist dadurch bedingt, dass im Testzeitraum eingebrachte Änderungen, wie die Aktualisierung der `NetworkConfiguration` und der Benutzertypen, immer wieder zum Zurücksetzten von Verbindungsinformationen geführt haben.

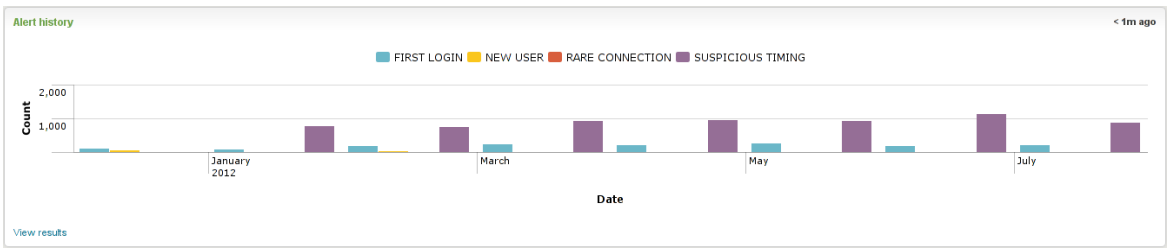

Abbildung 7.1.: Anzahl der einzelnen Alarme seit Inbetriebnahme von `LoginIDS` in `Splunk`

Insgesamt wurden dabei ungefähr 42.000 Zeilen `sshd`-Log und 12.000 Einträge aus dem Citrix-Windows-Log analysiert. Es wurden 8100 Alarme gemeldet. Das entspricht einer Reduktion der zu untersuchenden Log-Einträgen auf 15%. Der `LoginIDS`-Prototyp hatte in Tests vor dieser Arbeit eine Reduktion auf 9% erreicht [vEMH12a]. Erst nach einer längeren Phase ohne neue Änderungen können diese Werte verglichen werden.

Außerdem beeinflusst die Implementierung des Retentionsintervalls die Statistik, indem alte Benutzer und vor allem Verbindungen gelöscht werden, die bei erneuter Benutzung wieder einen Alarm auslösen. Diese Änderung führt allgemein zu mehr Alarmmeldungen. Um aussagekräftige Daten zu erhalten, muss der Testbetrieb mit der nach dieser Arbeit veröffentlichten stabilen `LoginIDS`-Version einige Zeit fortgesetzt werden. Eventuell müssen auch die Intervalle zum Löschen von Benutzern beziehungsweise Verbindungen angepasst werden.

7.1.2. Testsystem lokal

Auf dem lokalen System wurden ebenfalls `sshd`-Logs und zusätzlich Apache-Webserver-Logs analysiert. Dieses System diente hauptsächlich zu Tests, die die Kontinuität der Analyse stark einschränken würden. So wurde hier mehrfach eine neue Version sauber installiert und Updates von einer Version auf die nächste getestet, um die entsprechenden Anleitungen schreiben zu können und eine problemlose Integration in das System am LRZ zu gewährleisten.

Als System wurde hier ein Debian wheezy/sid, ein OpenSuse 12.1 und ein Ubuntu 12.04 eingesetzt. Alle von `LoginIDS` benötigten Perl-Module sind in diesen Linux-Distributionen in den jeweiligen Package-Repositories verfügbar. Außerdem wurde mit diesen Systemen auch die Weiterleitung mit `Splunk`-Forwardern beziehungsweise einer `MySQL`-Datenbank überprüft und getestet. Weitere Informationen zu den nötigen Paketen und der Installation befinden sich im Anhang A.

7.2. `LoginIDS` Setup mit `Splunk`-Forwarder

In diesem Abschnitt wird der Test von `LoginIDS` im Zusammenspiel mit einem `Splunk` „Universal forwarder“ beschrieben. Diese Installation basiert auf der `Splunk` Dokumentation zum Einrichten eines „Universal forwarders“ die unter `http://docs.splunk.com/Documentation/Splunk/latest/Deploy/Setupforwardingandreceiving` zu finden ist. In der Praxis ist das Forwarden der Ergebnisse der `LoginIDS`-Analyse relevant, da in der Implementierung am LRZ geplant ist, die Logdateien auf einem Host zu analysieren und die Ergebnisse auf einem zentralen `Splunk`-Server anzuzeigen.

Aus den in Kapitel 7.1.2 beschriebenen Systeme wurde für diesen Test das Debian System mit `Splunk` 4.3.1 unter der free-Licence und das openSUSE System mit dem `Splunk` „Universal forwarder“in der Version 4.3.3 verwendet.

Auf dem Debian System wurde die `Splunk` App installiert. Es muss bei einer neu Installation kein Setup durchgeführt werden. Die Input-Skripte werden nicht verwendet, da die Informa-

tionen von der anderen Splunk-Instanz übertragen werden. Zusätzlich muss aber das Empfangen von Forwardern eingeschaltet werden. Das geht unter dem Menüpunkt Manager → Forwarding and receiving → configure receiving → new. Damit ist diese System bereit.

Auf dem anderen System wird zuerst LoginIDS, wie im Anhang A beschrieben, installiert. Danach wird der Splunk „Universal forwarder“, wie in der Splunk-Dokumentation beschrieben, installiert. Da es im „Universal forwarder“ kein Web-Frontend gibt, muss die Installation der LoginIDS-App über das Kommandozeileninterface erfolgen. Im Ordner $SPLUNK_HOME/bin startet der Befehl „splunk install app loginIDS.spl“ die Installation. Dazu muss die LoginIDS-App ebenfalls in dem selben Ordner sein. Prinzipiell kann auch ein absoluter Pfad zur App angegeben werden. Die Konfiguration der Datenbank kann nun nicht mehr über das Splunk-Web-Interface durchgeführt werden, sondern muss manuell in der Datei loginIDS.conf vorgenommen werden. Diese Datei befindet sich im Ordner $SPLUNK_HOME/etc/apps/loginIDS/defaults und sollte zuerst in den Ordner $SPLUNK_HOME/etc/apps/loginIDS/local kopiert werden.

Nach der Anpassung der Datenbankverbindung, muss die Datei inputs.conf aus dem defaults-Ordner nach local kopiert werden und bei allen Skripten „disabled“ auf „0“ gesetzt werden. Zusätzlich kann das Intervall modifiziert werden. Zum Schluss muss noch angegeben werden, an welchen Server die Daten weitergeleitet werden sollen. Dies geht wieder aus dem Ordner $SPLUNK_HOME/bin mit dem Befehl „splunk add forward-server ip:port -auth admin:changeme“. Die Werte der Schlüsselwörter ip und port müssen dabei entsprechend ersetzt werden. Der Benutzername ist in diesem Fall admin und changeme das Passwort. Splunk verwendet den Port 9997 als Standard. Um die Änderungen zu übernehmen muss Splunk neugestartet werden. Dazu wird der Befehl splunk restart verwendet; dazu muss das Current-Working-Directory auf den bin-Ordner gesetzt sein. Jetzt werden Ereignisse von LoginIDS von einem Host auf den anderen weitergeleitet.

Unter Umständen muss noch eine neue Firewall-Regel erstellt werden, die den TCP-Netzwerkverkehr auf dem gewählten Port erlaubt.

8. Zusammenfassung und Ausblick

In dieser Arbeit wurden zur Optimierung der Erkennung von Innentätern zuerst verschiedene technische und organisatorische Maßnahmen betrachtet. Ausgehend von dem `LoginIDS`-Prototyp wurde im Laufe der Arbeit eine erweiterte Version erstellt.

`LoginIDS` untersucht Logdateien von verschiedenen System-Diensten auf Anomalien im Benutzerverhalten. Das Ziel ist dabei die Identifikation von Innentätern oder kompromittierten Accounts sowie die Reduktion auf wenige manuell zu prüfende Logeinträge. Als neues Quell-Logdatei-Format wurde das Logformat des Apache-Webservers zu den Formaten der `SSH`-Server-Logs und Citrix-Windows-Terminalserver-Logs hinzugefügt. Die Logdateien werden durch `LoginIDS` auf Veränderungen des normalen Benutzerverhalten überprüft. Hierbei werden veränderte Loginzeiten, Verbindungsdaten und die Häufigkeit der Verwendung einzelner Dienste überwacht. Zu dieser Analyse wird eine Datenbank verwendet, in der die Logeinträge zu Statistiken zusammengefasst werden. Dabei wird auch das geltende Datenschutzrecht beachtet, geeignete Retentionsintervalle definiert und durch automatisierte Löschverfahren umgesetzt. Die gefundenen Auffälligkeiten werden als Alarme gemeldet. Diese können in den neu eingeführten Trigger-Skripten behandelt und in dem Logfile-Management-System `Splunk` angezeigt werden.

Dabei wurde festgestellt, dass `LoginIDS` als eine Komponente zur Erweiterung eines vorhandenen Sicherheitssystems eingesetzt werden kann, um die manuell näher zu untersuchenden Log-Einträge deutlich zu reduzieren.

Neben der Erweiterung durch andere Dienst-Formate wäre es vorteilhaft eine eigene grafische Oberfläche für `LoginIDS` zu entwickeln und dadurch die Konfiguration durch einen Benutzer zu vereinfachen. Weitere Punkte, die zukünftig implementiert werden könnten, werden in Kapitel 8.2 beschrieben.

8.1. Diskussion

Die in der Aufgabenstellung geforderten und der Großteil, der sich zusätzlich in der Anforderungsanalyse in Kapitel 3 ergebenen Aufgaben, konnten umgesetzt werden. Von insgesamt 18 Anforderungen wurden die 13 wichtigsten implementiert. Die Tabelle in Kapitel 3.3 gibt eine Übersicht hierüber und verweist auf die entsprechenden Kapitel mit weiteren Erläuterungen.

Die Analysefähigkeit von `LoginIDS` wurde gesteigert, indem nun auch Apache-Webserver-Logs analysiert werden können. Zudem ist es durch das neue, modular gestaltete System, einfacher neue Konvertierungsskripte hinzuzufügen.

Die Reduktion der zu überprüfenden Log-Einträge auf 15%, wie sie in Kapitel 7.1.1 aufgeführt wurde, ist so noch nicht ausreichend. Weitere Tests werden zeigen, ob nach einer längeren Phase ohne grundlegender Änderungen am `LoginIDS`-System dieser Wert besser wird. Die Visualisierung der `LoginIDS`-Alarme wurde mit Hilfe von `Splunk` implementiert. Wie auf den Screenshots in Kapitel 6.2.2 zu sehen ist, wird die Ausgabe dadurch deutlich übersichtlicher dargestellt, als es in einer Textdatei möglich wäre. Die Implementierung von Triggern, wie sie in Kapitel 3.2.18 beschrieben wurden, ermöglicht aber trotzdem eine Ausgabe aller Alarme in eine Logdatei zur anderweitigen maschinellen Verarbeitung.

Die Visualisierung hat sich auch im Hinblick auf die Fehlererkennung bewährt. Bei einem Ausfall der Übertragung der Citrix-Windows-Logs wurde dies in der `LoginIDS`-App über die Anzahl der analysierten Dienste sofort ersichtlich.

`Splunk` bietet eine übersichtliche Darstellung und einfache Suche auf den von `LoginIDS` generierten Daten. Ein Problem besteht aber darin, dass es keinen Rückkanal von der `Splunk`-App an `LoginIDS` gibt. So kann zum Beispiel für Fehlalarme das `falsePositive`-Flag nicht direkt in `Splunk` gesetzt werden. Eine weitere Einschränkung ist, dass Tupel aus der Datenbank nur einmal importiert werden. Dadurch können veränderliche Felder wie der `lastSeen`-Zeitstempel in der `Splunk`-App nicht verwendet werden, da sie eventuell in der `LoginIDS`-Datenbank aktualisiert werden und damit die Datenbanken nicht mehr synchron sind.

8.2. Ausblick

In einer zukünftigen Arbeit könnten die im Folgenden beschriebenen Erweiterungen hinzugefügt werden, die im Rahmen dieser Arbeit nicht mehr umgesetzt werden konnten.

Für den Einsatz in einem unsicheren Netz müsste die Kommunikation zwischen `LoginIDS` und einer entfernten Datenbank beziehungsweise `Splunk`-Instanz abgesichert werden. Die Kommunikation wird momentan noch nicht verschlüsselt abgewickelt. Sowohl `MySQL` als auch das Forwarding mit `Splunk` unterstützen eine Verschlüsselung mit SSL. Ein zwischenzeitlicher Workaround könnte über einen `SSH`-Tunnel oder das Einbinden der `SQLite`-Datenbank über SSHFS sein. Der Wunsch, die Daten sicher zu übertragen, ist auch eine Weiterführung der in Kapitel 3.2.15 beschriebenen Integritätssicherung.

Mögliche Aufgaben ergeben sich aus den anderen, nicht weiter verfolgten Anforderungen, wie der Logout-Erkennung aus Kapitel 3.2.9 und der Aufzeichnung der gleichzeitig aktiven Verbindungen aus Kapitel 3.2.10. Die Speicherung der durchschnittlichen Verbindungszeit könnte dabei ein weiterer Faktor sein, um die Genauigkeit der Analyse von `LoginIDS` zu verbessern und dadurch die Anzahl der Alarmmeldungen weiter zu reduzieren. Dazu wäre es aber notwendig alle zu überwachenden Systeme mit einer neuen `SSH`-Version auszustatten.

Ebenso wäre es interessant, das in Kapitel 3.2.16 beschriebene Benutzermapping genauer zu untersuchen und festzustellen, wie viele Loginnamen ein und der selben Person zuzuordnen sind. Diese Anforderung könnte die Erkennungsleistung von `LoginIDS` weiter steigern.

Außerdem ist es in jedem Fall wünschenswert, die Anzahl der unterstützten Dienste weiter auszubauen. Zum Beispiel könnte `LoginIDS` auch zur Analyse von Logins auf einem Mailserver oder verschiedenen Webportalen am LRZ eingesetzt werden. Durch die Installation von geeigneter Logging-Softwäre wäre es auch möglich die in Kapitel 2.2 erwähnte Systemaufrufs-Analyse umzusetzen.

Um die Installation der `Splunk`-App zu vereinfachen, könnte das Skript, dass die Daten aus der `LoginIDS`-Datenbank holt in Python neu geschrieben werden. `Splunk` bringt in den meisten Varianten selber Python mit. Wenn dort schon die nötigen Module zum Datenbankzugriff enthalten sind, könnte auf die Installation der extra Perl-Module auf dem `Splunk`-Host verzichtet werden.

Davon unabhängig ist es geplant, die bestehende `Splunk`-App in die `Splunkbase`, den „App-Store“ für `Splunk`-Apps, einzustellen. Dazu wird der App-Ordner als `tar.gz` gepackt und in `loginids.spl` umbenannt. Dieses Archiv kann auf der `Splunkbase` hochgeladen werden. Nach einem Überprüfungsprozess durch `Splunk` wird die App dann in die `Splunkbase` übernommen.

In einem weiteren Schritt könnte ein eigenes Webinterface für `LoginIDS` entwickelt werden. Damit wäre es möglich, die Analyse nicht nur darzustellen, sondern auch einfach Änderungen an der Datenbank vorzunehmen und so die Konfiguration von `LoginIDS` einfacher zu gestalten.

Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DNS	Domain Name System
GPLv3	GNU General Public License Version 3
GSSAPI	Generic Security Services Application Program Interface
IDS	Intrusion Detection System
LDAP	Lightweight Directory Access Protocol
LRZ	Leibniz-Rechenzentrum
MWN	Münchner Wissenschaftsnetz
OSI	Open Systems Interconnection
PID	Prozess ID
SCP	Secure-Copy
SIEM	Security Information & Event Management
SQL	Structured Query Language
SSH	Secure-Shell
SSHFS	SSH Filesystem
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
VPN	Virtual Private Network
WLAN	Wireless Local Area Network

Abbildungsverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[BHH+11] Baker, Wade, Alexander Hutton, C David Hylender, Joseph Pamula, D Ph, Marc Spitler, Mark Goudie, Christopher Novak, Mike Rosen, Peter Tippett, and et al.: *2011 data breach investigations report*. Methodology, Band 36(Heft 5):1–65, 2011.

[Bun11] Bundesamt für Sicherheit in der Informationstechnik: *IT-Grundschutz Kataloge 12. Ergänzungslieferung - September 2011*. September 2011.

[Eik12] Eikenberg, Ronald: *Innenangreifer half bei Stuxnet-Infektion*, April 2012.

[Haa10] Haas, Sibylle: *Arbeitnehmer in Deutschland – Im Büro: Bespitzelt und überwacht*, November 2010.

[HMRvE12] Hommel, Wolfgang, Stefan Metzger, Helmut Reiser, and Felix von Eye: *Log file management compliance and insider threat detection at higher education institutions*, 2012.

[ISO05] ISO/IEC 27001:2005: *Information technology — Security techniques — Information security management systems — Requirements*. ISO/IEC, Geneva, Switzerland, 2005.

[KMV+10] Kandias, Miltiadis, Alexios Mylonas, Nikos Virvilis, Marianthi Theoharidou, and Dimitris Gritzalis: *An insider threat prediction model*. In Katsikas, Sokratis, Javier Lopez, and Miguel Soriano (editors): *Trust, Privacy and Security in Digital Business*, volume 6264 of *Lecture Notes in Computer Science*, pages 26–37. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-15152-1_3.

[Kuh10] Kuhr, Daniela: *Reaktion: Skandale bei Bahn, Telekom und Lidl – Schutz vor Bespitzelung am Arbeitsplatz*, März 2010.

[LÖ9] Löwer, Chris: *Bespitzelung – Mitarbeiter unter Generalverdacht*, Mai 2009.

[Lei11] Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften: *Jahresbericht 2011*, 2011.

[MHR11] Metzger, Stefan, Wolfgang Hommel und Helmut Reiser: *Migration gewachsener Umgebungen auf ein zentrales, datenschutzorientiertes Log-Management-System – Erfahrungen am Leibniz-Rechenzentrum*. In: *INFORMATIK 2011*, 2011.

[NS12] Net-Security.org: *It's official, corporate passwords are cheap*, April 2012.

[rsy09] *Platforms - rsyslog wiki*, Oktober 2009.

[Spi03] Spitzner, Lance: *Honeypots - definitions and value of honeypots*, May 2003.

[Spl12] Splunk Inc.: *Distributed deployment manual*, June 2012.

[vEMH12a] Eye, Felix von, Stefan Metzger und Wolfgang Hommel: *19. DFN Workshop „Sicherheit in vernetzten Systemen": Innentäter in Hochschulrechenzentren*. 2012.

[vEMH12b] Eye, Felix von, Stefan Metzger und Wolfgang Hommel: *Innentäter in Hochschulrechenzentren – organisatorische und technische Maßnahmen zur Prävention und Detektion*. In: Paulsen, Christian (Herausgeber): *Sicherheit in vernetzten Systemen: 19. DFN Workshop*, Seiten B–1 – B–19, Norderstedt, Januar 2012. 19. Workshop Sicherheit in vernetzten Systemen, Books on Demand.

Anhang

A. Installation von LoginIDS

Diese Anleitung basiert auf dem Installationsvorgang auf einem openSUSE 12.1 System. In anderen Linux-Distributionen ist der Vorgang ähnlich.

Um LoginIDS von anderen Benutzern zu trennen, wird empfohlen, einen eigenen LoginIDS-Benutzeraccount anzulegen. Im folgenden Beispiel wird der Benutzer „loginids“ genannt. Außerdem wird ein eigenes Verzeichnis /home/loginids für den Benutzer angelegt.

```
# useradd loginids -m -d /home/loginids
```

Jetzt kann die aktuellste Version unter https://git.lrz.de/?p=LoginIDS.git;a=tree gesucht und in dem neuen Benutzerverzeichnis gespeichert werden. Danach muss das Archiv entpackt werden.

```
# cd /home/loginids
# tar -xzvf LoginIDS_1.1.4.tar.gz
```

Bevor LoginIDS ausgeführt werden kann, müssen noch einige Perl-Module installiert werden.

```
# zypper install perl-Linux-Inotify2 perl-DateTime perl-Time-modules \
  perl-DBI perl-DBD-mysql perl-DBD-SQLite perl-Socket6 perl-File-Tail \
  perl-DateTime-Format-Strptime
```

Danach kann das Setup-Skript ausgeführt werden. Dort werden grundlegende Datenbankeinstellungen abgefragt.

```
# cd /home/loginids/LoginIDS
# ./setup.pl
```

Es besteht die Wahl SQLite oder MySQL als Datenbanksystem einzusetzen. Ohne weitere Konfiguration kann eine SQLite-Datenbank verwendet werden. Als Pfad kann dann zum Beispiel ./loginidsDB.sqlite angegeben werden. Dadurch wird die Datenbank in dem LoginIDS-Verzeichnis angelegt. Danach kann das Skript die Verbindung testen.

Anschließend muss das Skript noch die initialen Tabellen und Einträge erstellen.

Zum Schluss wird das Setup-Skript die Konfigurationswerte in die Datei loginids.conf schreiben. Weitere Konfigurationsoptionen sind in der Datei loginids.default.conf angegeben. Bei Bedarf können die veränderten Standartwerte in die Datei loginids.conf geschrieben werden.

Bevor `LoginIDS` gestartet werden kann, muss die Umgebungsvariable `LOGINIDS` definiert werden. Diese Variable enthält den vollständigen Pfad in das `LoginIDS`-Verzeichnis.

```
# export LOGINIDS=/home/loginids/LoginIDS
```

Jetzt kann `LoginIDS` gestartet werden.

```
# ./loginids start
```

Danach sollte `LoginIDS` laufen. Zu diesem Zeitpunkt werden aber noch keine Logdateien analysiert. Dies muss in einem nächsten Schritt eingerichtet werden. In dem `logs`-Ordner befinden sich alle von `LoginIDS` selber geschriebenen Logdateien.

B. Konfiguration von `LoginIDS`-Inputs

Dieser Abschnitt geht davon aus, dass `LoginIDS` bereits installiert ist. Die Konfiguration wird anhand der Logdateien des `sshd` `SSH`-Servers beschrieben. In openSUSE schreibt `sshd` seine Lognachrichten in die Datei `/var/log/messages`.

Der erste Schritt das Analysieren von Logdateien einzuschalten ist, das entsprechende Konvertierunsskript aus dem `converter-available` in den `converter-enabled` Ordner zu verlinken.

```
# cd /home/loginids/LoginIDS/converter-enabled
# ln -s ../converter-available/sshd_messages.pl messages
```

Damit wird `LoginIDS` dieses Konvertierungsskript für alle Dateien, die in den `logfiles/input` Ordner kopiert werden und deren Dateiname mit „messages“ beginnt, verwenden. Ein kompletter Dateiname muss immer den Logfiletyp, einen Hostnamen und einen Zeitstempel enthalten. Ein gültiger Dateiname wäre zum Beispiel „messages-localhost-2012-07-25“.

Um regelmäßig neue Logeinträge aus `/var/log/messages` in das `input`-Verzeichnis zu kopieren, wird hier das `tailFile.pl`-Skript aus dem `LoginIDS`-Ordner verwendet.

Das `tailFile`-Skript kann mehrere Dateien gleichzeitig auf neue Zeilen überwachen. Die neuen Zeilen werden in regelmäßigen Abständen in eine Datei, die sich in dem `input`-Ordner befindet, geschrieben. Jeder Parameter, der dem Skript übergeben wird, definiert die Datei, die überwacht werden soll, sowie den Typ der Datei und den zu verwendenden Hostnamen.

In diesem Beispiel heißt das Konvertierungsskript „messages“, also muss der Typ auch „messages“ sein. Der Hostname kann prinzipiell beliebig gewählt werden, darf aber kein „-“ enthalten. In diesem Fall wird „localhost“ verwendet. Das Zeichen „|“ trennt dabei das zu überwachende Logfile und die Typ, Hostname Bezeichnung.

```
# ./tailFile.pl '/var/log/messages|messages-localhost'
```

Das `tailFile`-Skript kann über die Datei `loginids.conf` weiter konfiguriert werden. Mögliche Parameter werden in der Datei `loginids.default.conf` beschrieben.

C. Installation der Splunk App

Die verschiedenen Optionen LoginIDS und Splunk zu installieren, sind in Kapitel 6.3 genauer beschrieben. Die Installation der App ist in jedem dieser Fälle gleich. Es wird hier davon ausgegangen, dass LoginIDS und die dazugehörige Datenbank schon eingerichtet wurden. Dieser Schritt wird in Anhang A beschrieben. Die App kann aus dem LoginIDS Git-Repository heruntergeladen werden. In dem Ordner https://git.lrz.de/?p=LoginIDS.git;a=tree;f=SplunkApp ist immer die aktuellste gepackte Version loginIDS.spl zu finden. Diese Datei kann in Splunk über Manager → Apps → Install app from file installiert werden.

Bei der Installation müssen noch einige Einstellungen vorgenommen werden. Den Setup-Dialog erreicht man über das Splunk-Frontend, indem man unter Manager → Apps in der Liste die Zeile der LoginIDS-App sucht und dort die Aktion Set up ausführt. Diese Einstellungen beziehen sich hauptsächlich darauf, wie die LoginIDS-Datenbankanbindung geschehen soll und wie oft die Datenbank auf neue Ereignisse hin abgefragt werden soll. Im Auslieferungszustand der App sind alle Inputs deaktiviert. Es muss auch zuerst eine Datenbankverbindung konfiguriert werden, bevor die Inputs aktiviert werden. Abbildung C.1 zeigt den Setup-Dialog in Splunk.

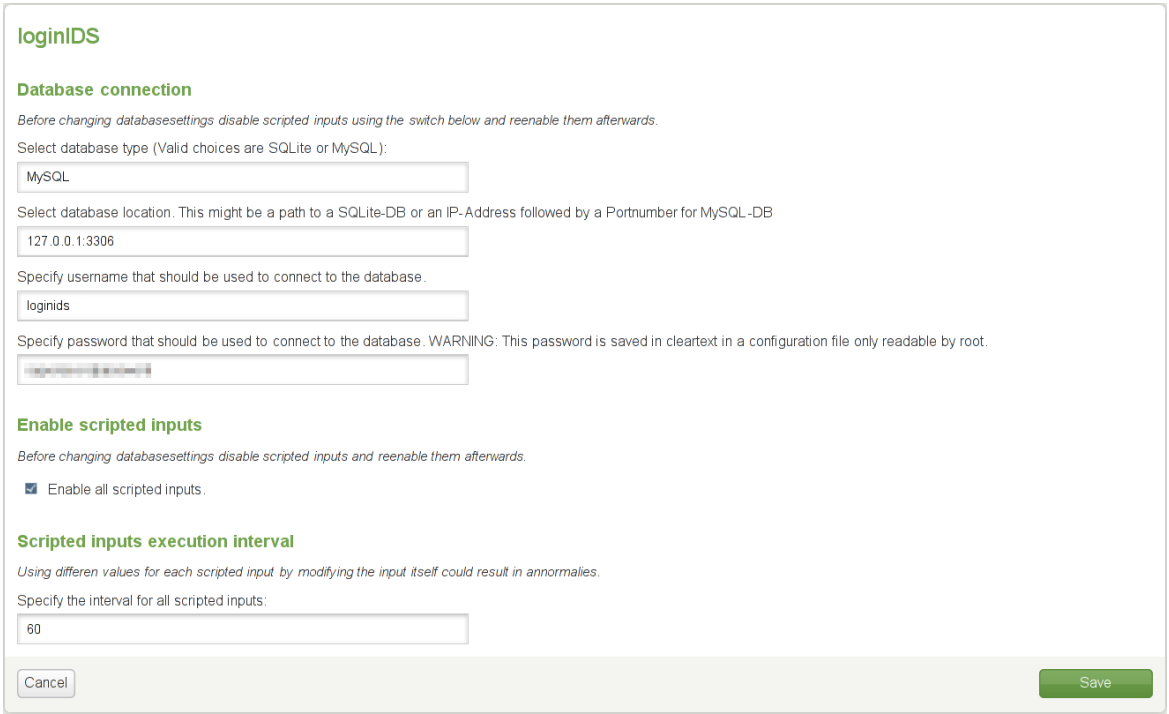

Abbildung C.1.: Der Setup-Dialog der LoginIDS-Splunk-App

D. Entfernen der `LoginIDS-Splunk`-App

Um eine App aus `Splunk` zu entfernen, gibt es zwei Methoden, zu denen root-Rechte benötigt werden. Die bevorzugte Methode ist es, den Befehl „`splunk remove app loginIDS`“ auszuführen. Das Programm `splunk` befindet sich in `$SPLUNK_HOME/bin/`. Alternativ kann das Verzeichnis von `LoginIDS`, das unter `$SPLUNK_HOME/etc/apps/` angelegt wurde, gelöscht werden.

Danksagung

Für die Unterstützung mit hilfreichen Kommentaren und die ständige Diskussionsbereitschaft bedanke ich mich bei meinen Betreuern Felix von Eye, Dr. Wolfgang Hommel und Stefan Metzger.