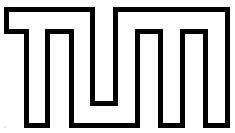


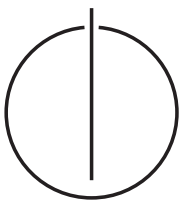
Technische Universität München

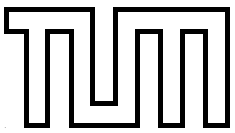
Fakultät für Informatik

Systementwicklungsprojekt (SEP)

Entwurf und Implementierung eines metrikbasierten Reporting-Systems für IT-Sicherheit

Johann Schlamp





Technische Universität München

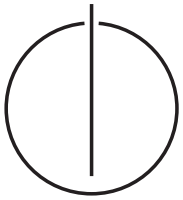
Fakultät für Informatik

Systementwicklungsprojekt (SEP)

Entwurf und Implementierung eines metrikbasierten Reporting-Systems für IT-Sicherheit

Design and Implementation of a metric-based reporting system for IT-security

Bearbeiter:	Johann Schlamp
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Nils gentschen Felde Matthias Hofherr (atsec information security GmbH)
Abgabedatum:	15. Januar 2009



Abstract

Gewöhnliche Information Security Management Systeme legen ihren Fokus auf das Sammeln von sicherheitsrelevanten Daten oder beschränken sich auf die starre Darstellung von Sicherheitszuständen zu einem gegebenen Zeitpunkt. Da IT-Sicherheit aber als Prozess verstanden werden muss, reicht ein System zur Ermittlung eines augenblicklichen Zustandes nicht aus. Auch bei Umsetzung der zyklischen Ansätze diverser Standards für Datensicherheit („plan-do-check-act“) wird oft mehr Wert auf das Lösen konkreter Probleme gelegt als auf kontinuierliche Kontrolle von Fortschritt und Verbesserung. Das bloße Sammeln von Daten sowie stationäre Check & Act Vorgehensweisen sollten durch „echtes“ Messen ergänzt werden, das Vergleiche und Bewertungen zulässt, strategisches Handeln erlaubt und Entwicklungen deutlich erkennbar macht.

Ziel dieses Systementwicklungsprojektes ist es, die Grundlage für ein intuitives Reporting-System zu schaffen, das den obigen Anforderungen entsprechen kann. Dazu müssen Sicherheitsereignisse und -informationen aller Art zusammengeführt, sinnvoll korreliert und mit Hilfe von geeigneten Metriken gemessen werden. Es werden prinzipielle Ansätze zu Korrelationen und Metriken erarbeitet sowie die konkret benutzte Infrastruktur erläutert. Hauptaugenmerk liegt auf der Behandlung der Security Events mit resultierenden Alerts sowie deren Darstellung in einer GUI.

Inhaltsverzeichnis

1 Problemstellung

Mit Einzug der Informationstechnologie in Firmen, Behörden und private Haushalte musste der Begriff von *Sicherheit* neu definiert werden. Physikalischer Schutz wertvoller Objekte ist weiterhin wichtig, aber moderne Angriffe zielen oft auf die Kompromittierung von Rechensystemen und damit auf Verletzung der Vertraulichkeit, Verfügbarkeit oder Integrität von Daten ab. Informationen sind in unserer Zeit wertvoller denn je, daher sollte deren Schutz als Disziplin der IT-Sicherheit hohe Priorität einnehmen.

Mit zunehmender Vernetzung und Anbindung an das Internet werden Angriffspunkte geschaffen, die unsichtbare Angriffe ermöglichen und damit zu Schäden führen können, deren Ausmaß oft erst Monate später erkennbar ist. Es sind also nicht nur technische Maßnahmen zur Erhaltung der Sicherheit von Bedeutung, sondern ebenso Verfahren zur frühzeitigen Erkennung von Einbrüchen oder Einbruchsversuchen.

Um ein Bewusstsein für die konkrete Gefährdung und eventuell vorhandenen Schwachstellen zu schaffen ist es weiterhin nötig, der entscheidungstragenden Instanz für Sicherheit leicht verständliche und nicht-technische Informationen zur Verfügung zu stellen. Diese sollten intuitiv und abstrahierend sein, aber dennoch aussagekräftig und informativ genug, um Beschlüsse in sicherheitsrelevanten Fragen stützen zu können.

Die vorliegende Arbeit nimmt sich als Systementwicklungsprojekt zum Ziel, den Prototyp eines Reporting-Systems am Beispiel eines kleinen Firmennetzes zu realisieren. Der zentrale Bestandteil dabei ist eine Einbruchserkennung, die auf der Untersuchung von Beziehungen zwischen Ereignissen beruht. Hierbei wird großer Wert auf eine generische Implementierung gelegt, um einen Einsatz in anderen Szenarien wie dem Münchner Wissenschaftsnetz MWN, aber auch in einem privaten Heimnetz grundsätzlich zu gewährleisten.

1.1 Lösungsidee

Die zugrunde liegende Lösungsidee basiert auf Einbruchserkennung durch *Korrelation*. Dazu sind zwei Schritte notwendig, die durch einen verteilten Aufbau des Systems reflektiert werden können. Im ersten Teil werden aus verschiedenen Quellen relevante Informationen zusammengetragen und in Echtzeit an das zweite Teilsystem weitergeleitet. Diese werden dort korreliert, also auf Gemeinsamkeiten oder Zusammenhänge überprüft. Finden sich mehrere Hinweise auf einen Angriff, die zueinander in Beziehung stehen, so wird ein Alarm erzeugt. Beim Eintreffen weiterer korrespondierender Indizien steigt die Wahrscheinlichkeit eines Einbruchs und dementsprechend auch die Priorität des Alarms.

Es wird nicht nur nach technischen Details korreliert, sondern auch nach „semantischen Metainformationen“, die durch eine vorangehende Kategorisierung jedem Ereignis zugeteilt werden. Dadurch lassen sich auch Beziehungen in scheinbar zusammenhangslosen Vorfällen erkennen.

Abschnitt 2.2 *„Einordnung dieser Arbeit“* und insbesondere Abschnitt 4.3 *„Filterung und Parsing“* sowie Abschnitt 4.4 *„Korrelation“* beschreiben detailliert die vorgestellten Ideen.

1.2 Struktur der Arbeit

Im Anschluss an diese Einführung folgt die konkrete Ausarbeitung des Projekts. Dabei liegt folgende Struktur zu Grunde:

Kapitel 2 erläutert die Grundlagen des IDS. Zuerst wird dabei auf unterschiedliche Ansätze der Einbruchserkennung eingegangen. Anschließend folgt die Einordnung dieser Arbeit.

Kapitel 3 legt den konkreten Aufbaus des Systems fest. Es werden Anforderungen definiert sowie die notwendigen Komponenten erörtert. Eine genaue Spezifikation der Implementierung folgt im nächsten Abschnitt.

Kapitel 4 ist neben dem Systemdesign der Hauptteil der Arbeit. Hier werden die Grundlagen der Eventbehandlung einschließlich Kategorisierung, Priorisierung und Korrelation erörtert sowie die verwendeten Tools und die konkrete Implementierung dokumentiert. Außerdem werden die Funktionen der GUIs vorgestellt.

In **Kapitel 5** folgt eine Beschreibung verschiedener Testszenarien sowie eine Auswertung der Ergebnisse.

Abschließend gibt **Kapitel 6** einen zusammenfassenden Überblick und eine perspektivische Sicht auf mögliche Erweiterungen.

Im Anhang findet sich das Abbildungs- und Literaturverzeichnis.

2 Grundlagen

Dieses Kapitel stellt unterschiedliche Ansätze und differenzierte Analysemethoden für eine Einbruchserkennung vor. Im Anschluss daran wird die Lösungsidee dieser Arbeit konkretisiert und in bereits vorhandene IDS-Ansätze eingeordnet.

2.1 Verschiedene IDS-Ansätze

Bei näherer Untersuchung etablierter Lösungen ist festzustellen, dass sich die Definition des Begriffs *IDS* in zwei Teile gliedern lässt. Entscheidend dabei ist der Kontext, in dem die Bezeichnung verwendet wird. *James F. Kurose* und *Keith W. Ross* definieren in ihrem Buch *Computer networking* ein IDS wie folgt:

> „*A device that generates alerts when it observes potentially malicious traffic is called an* **Intrusion Detection System (IDS)**." [Kurose and Ross, 2007, S.771]

Aus dem Blickwinkel eines zu schützenden Netzes heraus macht diese Definition durchaus Sinn; Anwendungsgebiete wie Überwachung der Datenintegrität auf Großrechnern dagegen werden von dieser Interpretation nicht abgedeckt. *Michael Meier* führt dazu die Unterscheidung eines **Host-basierten IDS** ein, das „*Ereignisse auf Betriebssystem- oder Anwendungsebene, z.B. Systemrufe, verarbeitet*" [Meier, 2007, Kapitel 3].

In den nächsten Abschnitten werden Vor- und Nachteile des jeweiligen Ansatzes sowie verschiedene Arten der Analyse erörtert, aus denen anschließend die Zielsetzung dieser Arbeit nochmals begründet wird.

2.1.1 Netz-basiertes IDS

Ein Netz-basiertes IDS beruht grundsätzlich auf der Analyse von Netz-Paketen. Dabei wird der Inhalt von Paketen auf bekannte *Signaturen* oder auffällige Verhaltensmuster (siehe Unterabschnitt 2.1.3) untersucht. Dementsprechend kann eine Vielzahl von Angriffen erkannt werden, zu denen Portscans, Schwachstellen-Angriffe (sog. *Exploits*) auf Anwendungen und Betriebssysteme sowie Würmer und Viren zählen. Ein bekannter Vertreter dieser IDS-Art ist sicherlich *Snort*, auf das in Unterabschnitt 4.1.2 näher eingegangen wird.

Da viele Netze Verkehr im Gigabit-Bereich aufweisen, kann eine derartige Paket-basierte Verarbeitung sehr aufwendig werden. Daher wird hier meist mit Paket-Puffern gearbeitet, die eine vollständige Abarbeitung aller Pakete sicherstellen. Abhilfe können hier auch mehrere Sensoren schaffen, die sich die Last teilen. Aber auch aus einem anderen Grund ist dies häufig notwendig: Oft sind Netze segmentiert, wodurch eine direkte Kommunikation von Endgeräten nur innerhalb eines solchen Segments möglich ist. Andere Netzteile können nur über *Router* oder *Bridges* erreicht werden, die im Bedarfsfall als Schnittstelle fungieren. Da dadurch nicht mehr der gesamte Ethernet-Verkehr an einem Endpunkt anliegt, sondern

nur noch der Teil des entsprechenden Netz-Segments, wird ein Sensor für jede topologische Einheit erforderlich.

Trotz sorgfältiger Aufstellung von Sensoren kann eine Überbelastung des Netzes zu Spitzenzeiten oder während eines Angriffes das IDS dennoch unzuverlässig machen. Wird der Verkehr auf den Sensoren nicht gepuffert, so können Pakete bei der Analyse verloren gehen. Sofern der Anstieg des Netzverkehrs nicht zu drastisch wird, stellen Puffer zwar die korrekte Analyse aller Pakete sicher, dennoch leidet darunter aber die Echtzeit- Fähigkeit. Daher ist es möglicherweise sinnvoll, den zu untersuchenden Verkehr einzuschränken. Unter der Annahme, dass kein Angreifer sich physikalisch Zugang zum Netz verschaffen kann, reicht eine Analyse der eingehenden Pakete an *Firewalls* aus. Wird allerdings ein entsprechender Zugang kompromittiert, ist es nicht mehr möglich, den Angreifer weiter zu verfolgen.

Wichtigster Vorteil und Einsatzgrund von Netz-basierter Einbruchserkennung ist aber dennoch die breite Abdeckung sowie hohe Ausfallsicherheit. Im Gegensatz zu einem lokalen, Host-basierten IDS führt der Ausfall eines einzelnen Systems nicht zur Lahmlegung des gesamten IDS. Vor allem das Zusammenspiel von mehreren Sensoren ermöglicht eine flächendeckende und ausfallsichere Überwachung eines Netzes.

2.1.2 Host-basiertes IDS

Ein weiterer Ansatz von Einbruchserkennung zielt nicht auf die Überwachung von Netzverkehr ab, sondern auf die Beobachtung lokaler Vorgänge auf bedeutsamen Systemen. Dort stehen in der Regel sehr viele Informationen wie Log-Dateien sowie Kernel- und Betriebssystem-Daten zur Verfügung. Außerdem können sog. *Integrity Checker* verwendet werden, die jegliche unzulässige Veränderung am Dateisystem erkennen können.

Fasst man diese Informationen zusammen, so entsteht eine nahezu lückenlose Überwachung des betreffenden Systems. Dadurch kann im Falle eines Angriffes auf ein sehr detailliertes Profil des Hergangs zurückgegriffen werden und dementsprechend eine differenzierte Behandlung folgen. Gibt es nur einige wenige Systeme mit sicherheitsrelevanten Daten und Diensten, so ist der Einsatz eines Host-basierten IDS zweckmäßig.

Wesentlicher Nachteil ist allerdings die Anfälligkeit. Wird ein entsprechendes System kompromittiert, ist die erste Aktion meist das Deaktivieren des IDS. Auch wenn möglicherweise einige Komponenten des IDS vor einem Angreifer versteckt werden können (wie z.B. der Integrity Checker *Samhain* [http://www.la-samhna.de/samhain/index.html]), so liegt die Anfälligkeit dennoch schon in der Struktur des IDS als prinzipieller Teil des Angriffsziels.

Für entscheidende Systeme wie Server oder Großrechner ist ein Host-basiertes IDS aber immer die erste Wahl. Im Falle eines Angriffes auf vertrauliche Daten müssen präzise Aussagen über das Ausmaß der Schäden getroffen werden können. Sollte tatsächlich das IDS selbst ausfallen, so wird der *worst case* angenommen.

2.1.3 Signatur- oder Anomalieerkennung

Grundsätzlich gibt es zwei verschiedene Analysemethoden, um bereitgestellte Informationen auf Einbrüche untersuchen zu können. Als erstes ist die *Signatur-basierte* Analyse anzuführen, bei der Ereignisse mit bereits bekannten Angriffsmustern verglichen werden. Da diese Signaturen simple Strings darstellen, mit denen ein *Pattern Matching* durchgeführt wird, ist diese Analyseart relativ effizient und leicht erweiterbar. Dennoch beschreibt dieser Umstand zugleich auch dessen Nachteil. Neuartige Angriffe, für die noch keine Signaturen

vorhanden sind, können nicht erkannt werden, wodurch bis zum nächsten Update prinzipiell immer Sicherheitslücken vorhanden sind. Außerdem ist eine stetige Pflege und Wartung des Systems unabdingbar.

Diesem *Blacklisting*-Prinzip, bei dem nur konkret definierte Angriffe erkannt werden können, steht die *Anomalieerkennung* gegenüber. Hierbei werden Ereignisse nicht mit Angriffsmustern verglichen, sondern mit der „normalen Ausgangssituation“. Alles, was sich davon unterscheidet, wird als verdächtig behandelt. Da ein „üblicher“ Netzverkehr aber kaum zu definieren ist, und auch vorangehende Kalibrierungs- und Messphasen nicht alle grundsätzlich erlaubten Ereignisse beinhalten werden, ist diese Art der Analyse nicht nur schwierig zu implementieren, sondern auch strukturell anfällig für *False Positives*.

Ebenfalls zur Anomalieerkennung zählen *statistische Analysen* und *Schwellwertanalysen*.

2.2 Einordnung dieser Arbeit

Um möglichst viele der aufgezeigten Vorteile auszunutzen und die Auswirkungen der entsprechenden Nachteile zu verringern, wird im Rahmen dieses Projekts ein *hybrider* Ansatz gewählt. Abbildung 2.1 verdeutlicht das Vorgehen.

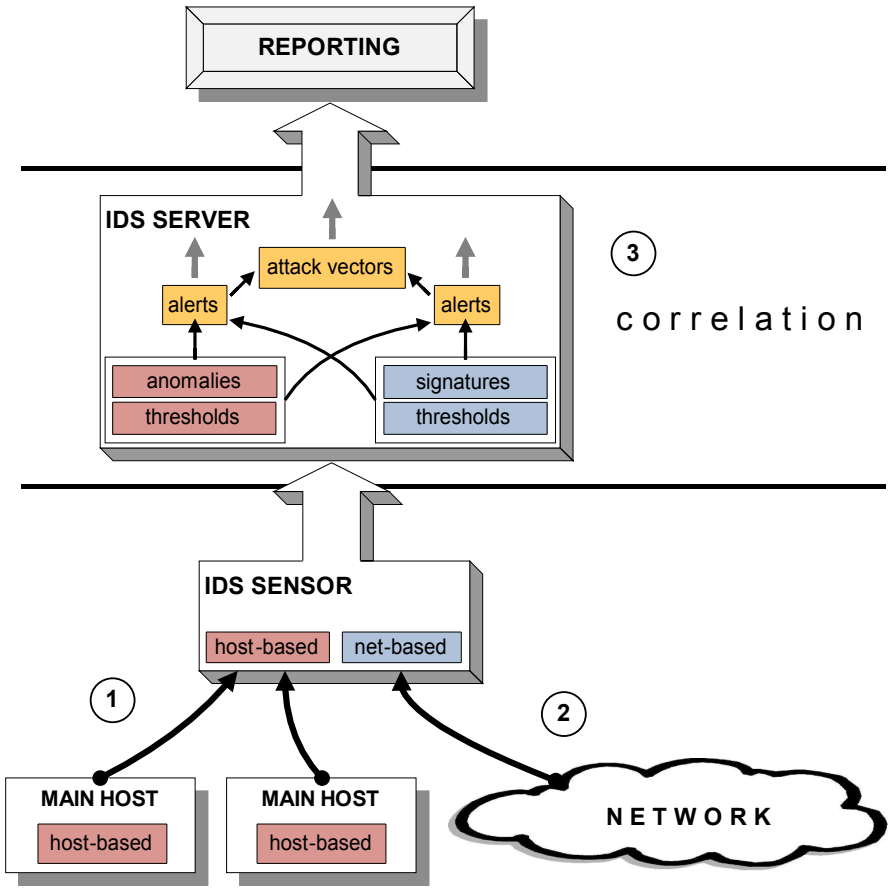


Abbildung 2.1: Hybrider IDS-Ansatz

① Auf allen sicherheitskritischen Systemen (*„Main Hosts“*) wird ein lokales Host-basiertes IDS implementiert, das Ereignisse direkt an den *IDS Sensor* weiterleitet. Dieser sendet alle relevanten Meldungen unbearbeitet und instantan an den *IDS Server*.

② Jeder Sensor verfügt über ein Netz-basiertes IDS, das auf Paket-Ebene nach sicherheitsrelevanten Ereignissen sucht. Die daraus entstehenden Meldungen sowie zusätzliche Log-Dateien von Firewalls, die dem Sensor unausgewertet zur Verfügung gestellt werden, werden ebenfalls direkt an den IDS Server weitergeleitet.

③ Hier findet die eigentliche korrelationsbasierte Analyse statt. Alle eintreffenden Meldungen, die demselben Vorfall zugeordnet werden können, führen ab einem bestimmten Schwellwert zu einem Alarm. Entscheidend hierbei ist, dass die Ereignisse unabhängig von ihrer Herkunft korreliert werden.

Events aus ① stellen unübliche Vorgänge (Anomalien) auf bestimmten Hosts dar, wohingegen bei Ereignissen aus ② bekannte Angriffsmuster im Netzverkehr entdeckt wurden. Diese Informationen werden durch Korrelation auf Gemeinsamkeiten oder Zusammenhänge untersucht und führen je nach Anzahl und Priorität der jeweiligen Einzelereignisse zu unterschiedlich kritischen Alarmen. Deren *Kritikalität* wird bei eintreffenden Folge-Events entsprechend erhöht.
In einem zusätzlichen Schritt wird auf Zusammenhänge in beliebigen bereits generierten Alarmen geprüft, um sog. *Angriffsvektoren* zu erkennen.

Das nächste Kapitel schildert die Anforderungen an das gesamte System und beschreibt die dazu notwendigen Komponenten. Insbesondere wird der oben dargestellte hybride IDS-Ansatz in Unterabschnitt 3.1.4 konkretisiert.

3 Systemdesign

In Abschnitt 2.2 wurde der grundlegende Ansatz für die verwendete Einbruchserkennung festgelegt (siehe Tabelle 3.1). Der folgende Abschnitt stellt eine Übersicht der gegebenen Anforderungen dar, die zu einer genauen Beschreibung der notwendigen Komponenten führt.

3.1 Anforderungen

Diese Anforderungen wurden direkt aus dem hybriden IDS-Ansatz (siehe Abschnitt 2.2) abgeleitet und sind als gegeben zu betrachten.

3.1.1 Erfassen relevanter Informationen

In komplexen Computernetzen gibt es meist eine Vielzahl unterschiedlicher Möglichkeiten, sich mit dem Netz zu verbinden. Dazu zählen Switches und Access Points zur physikalischen Anbindung sowie *remote-Zugänge* wie ssh, VPN, Mail- und Chatserver und viele weitere mehr. Hinzu kommen nutzerbeschränkte Dienste, die vom Internet aus erreichbar sein müssen. Dabei handelt es sich beispielsweise um geschützte Webservices, die vertrauliche Informationen wie Terminkalender, Zeiterfassung, Wikis und andere Interna über öffentliche Zugänge zur Verfügung stellen. Da Angriffe oft auf Schwachstellen solcher Dienste abzielen, können diese im Falle eines Einbruchs wertvolle Hinweise auf den Angreifer liefern und sollten daher in eine Erkennung miteinbezogen werden.

Weiterhin müssen lokale Fehlermeldungen berücksichtigt werden, die auf eine Kompromittierung oder einen Angriff aus dem Inneren heraus schließen lassen. Dazu zählen *Privilege Escalations* ebenso wie das Starten oder Stoppen von Diensten sowie andere verdächtige interne Ereignisse wie das Wechseln einer MAC-Addresse oder das Auftauchen eines neuen Betriebssystems. Auch diese Informationen können relevante Hinweise auf einen Einbruch liefern.

Folglich muss der erste Schritt zur Einbruchserkennung ein sinnvolles Auswählen verwertbarer Informationen sein. Um dem generischen Anspruch des Projekts gerecht zu werden, müssen allgemeine Angriffsszenarien betrachtet und die Informationen dementsprechend ausgewählt werden. In diesem Zusammenhang werden Ereignisse dann in ein einheitliches Format gebracht, um eine Weiterverarbeitung im nächsten Schritt zu ermöglichen (siehe Abschnitt 4.3).

3.1.2 Erweiterung durch Korrelation

Auch wenn die Menge der Informationen bereits durch Eingrenzung auf sicherheitsrelevante Ereignisse eingeschränkt wurde, bleibt sie in einem größeren Netz dennoch unüberschaubar. Nur durch eine weitere Reduzierung kann verhindert werden, dass eine Überzahl von Alarmen generiert wird. Die Begrenzung von *False Positives* und *False Negatives* ist ebenfalls ein wichtiger Gesichtspunkt. Deren Verhältnis zur Anzahl der tatsächlichen sicherheitskritischen

Vorfälle sollte immer überprüft werden, um die Nutzbarkeit des *Intrusion Detection Systems (IDS)* aufrecht zu erhalten (siehe Abschnitt 4.4).

Priorisierung nach Häufigkeit

Eine erste Möglichkeit der Verminderung von *Alerts* ist die Zusammenfassung gleicher oder ähnlicher Ereignisse. Erst bei der Überschreitung eines Schwellwerts wird ein Alarm erzeugt. Zusätzlich müssen diese Alerts je nach Anzahl und Typ der Ereignisse unterschiedliche Prioritäten aufweisen. Das klassische Beispiel hierfür sind fehlgeschlagene *Remote Logins*: ein falsch eingegebenes Passwort passiert häufig und ist dementsprechend unkritisch. Zehn fehlgeschlagene Logins dagegen sind verdächtig und sollten zu einem Alert führen, wohingegen Zehntausend Fehler mit großer Wahrscheinlichkeit einen *Wörterbuchangriff* darstellen. Es liegt auf der Hand, dass diese unterschiedlichen Vorfälle unterschiedlich priorisiert und gemeldet werden müssen.

Erkennung von Angriffsvektoren

Der eigentliche Vorteil einer korellationsbasierten Einbruchserkennung liegt aber auf der Wahrnehmung von Angriffen, die durch alleiniges manuelles Auswerten von Logdateien nicht entdeckt werden können. Dazu zählen sowohl verteilte als auch mehrstufige Angriffe. Hier können ebenfalls Remote Logins als Beispiel angeführt werden: oftmals beschränken sich Wörterbuchangriffe auf einige wenige Standard-Benutzernamen wie *test* oder *admin*. Wurden diese möglichen Standard-Konten nicht deaktiviert, kann sich ein Angreifer durch flächendeckendes Testen eines kompletten Netzes, aber mit nur sehr wenigen Versuchen auf einzelnen Maschinen leicht Zugang verschaffen. Durch Korrelation kann ein solches Angriffsmuster in Echtzeit erkannt werden, wozu normalerweise die Koordination mehrerer Systemadministratoren und ein aufwendiges Beobachten von Log-Dateien nötig wäre.

Weiterhin kann aus einem *Initialereignis* mit niedriger Priorität noch nicht auf einen umfassenden Einbruch geschlossen werden. Dazu sind vielmehr Folge-Events notwendig, die in Zusammenhang zum ursprünglichen Ereignis stehen. Die Erkennung dieser mithin als *Angrifssvektoren* bezeichneten Verkettungen soll technisch ebenfalls umgesetzt werden. Beispielhaft für diese Szenarien sind erkannte Portscans, die an sich zwar schon Aufmerksamkeit erregen müssen, aber aus denen noch nicht auf einen erfolgreichen Einbruch geschlossen werden kann. Erst wenn ein Hinweis auf lokale Kompromittierung der entsprechenden Zielmaschine folgt (wie beispielsweise das Abschalten des Log-Mechanismus), kann von einem effektiven Angriff ausgegangen werden.

Um diese Art der Einbruchserkennung zu realisieren, ist eine Kategorisierung der verschiedenen Ereignisse notwendig. Eine detaillierte Beschreibung dieses Vorgehens findet sich in Abschnitt 4.3.

3.1.3 Einsatz von Metriken

Ein „gedächtnisloses“ IDS, das nur im akuten Angriffsfall alarmiert, aber keine Informationen über frühere Vorfälle bereitstellt, ist allerdings nur bedingt sinnvoll. Standards für IT-Sicherheit wie das **IT-Grundschutzhandbuch** [BSI, 2008] des Bundesamts für Sicherheit in der Informationstechnik oder **ISO 27001** [ISO, 2005] fordern das regelmäßige Messen und Verbessern des Sicherheitszustandes. Im Zuge eines *plan - do - check - act* Kreislaufs (siehe Abbildung 3.1) muss daher die Effektivität der Einbruchserkennung kontinuierlich

überprüft werden. Dieser Zyklus beschreibt allgemein die Phasen eines stetigen Verbesserungsprozesses: Nach der Planung eines Prozesses wird dieser prototypisch realisiert und intensiv auf die Erreichung des Ziels überprüft. Die dabei erkannten nötigen Verbesserungen liefern den Ausgangspunkt für den nächsten Zyklus.

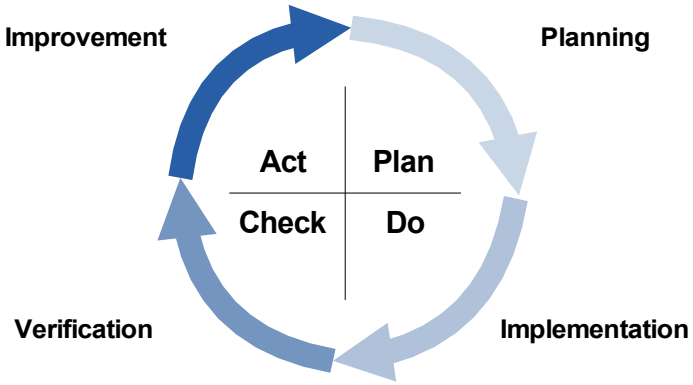


Abbildung 3.1: PDCA-Zyklus

Angewandt auf IT-Sicherheit fordert dieser Kreislauf das kontinuierliche Überwachen des Sicherheitszustandes. Dazu eignen sich sogenannte *Metriken*, die – sinnvoll definiert – konkrete Aussagen über die Gefährdungslage liefern können. Ein einfaches Beispiel für eine Metrik ist ein *Zeitserien-Diagramm* (siehe Abbildung 3.2), das die monatliche Anzahl von Angriffsversuchen darstellt. Damit lassen sich Tendenzen bzw. Handlungsbedarf klar erkennen. Setzt man diese Werte zudem in Beziehung zu den tatsächlich erfolgreichen Einbrüchen, so ergibt sich ein aussagekräftiges Maß für die konkrete Sicherheitslage.

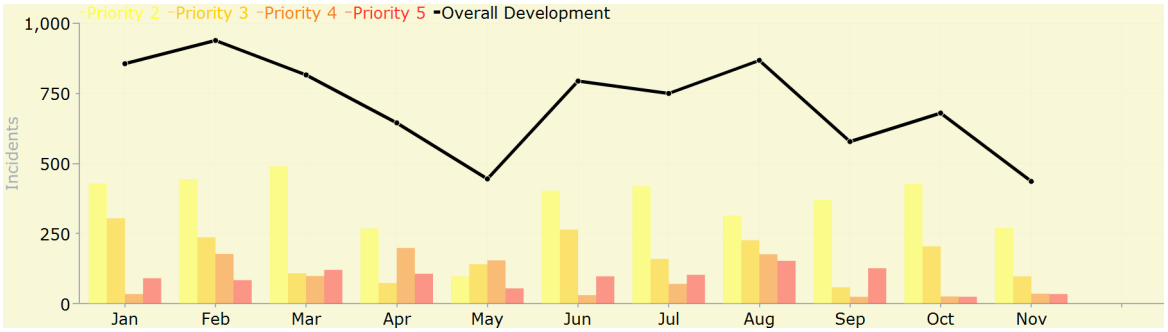


Abbildung 3.2: Beispiel für Zeitserien-Diagramme

Derartige Maßzahlen eignen sich nicht nur für die Kontrolle und Verbesserung der Einbruchserkennung, sondern können auch finanzielle Investitionen reflektieren und rechtfertigen. Im obigen Diagramm (Abbildung 3.2) könnte beispielsweise im Februar eine Sicherheitsstrategie beschlossen worden sein, die bis Mai vollständig umgesetzt wurde und damit dort einen Tiefpunkt der Vorfälle darstellt. Neuartige Bedrohungen führen zu einem erneuten Anstieg, der sich ebenso wie nachfolgende Gegenmaßnahmen im Diagramm widergespiegelt.

Desweiteren wird ein System zur Bewertung der eigenen Sicherheitslage aber auch in *security audits* und von Wirtschaftsprüfern verlangt, um ein Bewusstsein über die potentielle Gefahr und das damit einhergehende verantwortungsbewußte Handeln zu belegen.

Natürlicherweise kann das Systementwicklungsprojekt diesem Anspruch nur exemplarisch gerecht werden, allerdings wird auch hier Wert gelegt auf eine generische Implementierung mit Hinblick auf zukünftiger Erweiterung.

3.1.4 Resultierender Ansatz

Aus den oben geschilderten Anforderungen sowie dem in Abschnitt 2.2 beschriebenen hybriden IDS-Ansatz lässt sich der für diese Arbeit konkret eingesetzte Systemaufbau ableiten. Tabelle 3.1 definiert notwendige Komponenten und legt deren grundsätzlichen Aufstellung fest. Dabei ist zu bemerken, dass für diese Arbeit Netze immer als segmentiert betrachtet werden. Ist dies in einem konkreten Anwendungsfall nicht gegeben, so kann dieser Ansatz unter der Annahme eines einzigen Segments weiterhin angewandt werden.

Eine detaillierte Beschreibung der einzelnen Komponenten folgt im Anschluss. Abbildung 3.3 verdeutlicht den Aufbau des Systems mit Hilfe eines *UML Deployment Diagramms*.

Jedes Netz-Segment wird mit einem *Sensor* zur Netz-basierten Einbruchserkennung ausgestattet. Dieser empfängt zudem alle Log-Dateien der entsprechenden *Firewalls*. Auf sicherheitskritischen Systemen wie Server oder Großrechner wird ein lokales IDS implementiert, das Ereignisse unausgewertet an den zugehörigen Sensor weiterleitet. Alle Sensoren kommunizieren mit dem zentralen *IDS Server*, der vorgefilterte Ereignisse in Echtzeit erhält. Auf diesem Server erfolgt das Parsen und Korrelieren der *Events* sowie die Verwaltung und Darstellung der *Alerts*.

Tabelle 3.1: IDS-Ansatz

3.2 Komponenten

Der erarbeitete Ansatz (Tabelle 3.1) definiert im Wesentlichen die zwei Kern-Komponenten *Server* und *Sensor*. Weitere Informationsquellen wie zusätzliche Host-bezogene Logs von Datenservern sowie Firewall-Ereignisse dagegen bleiben transparent und können bei der Auflistung der zentralen Komponenten außen vor gelassen werden. Vielmehr sind sie als Teil der Sensoren (und möglicherweise auch des Servers) aufzufassen und werden in den nächsten beiden Abschnitten am Rande mit in die Betrachtungen aufgenommen.

Das folgende Diagramm (Abbildung 3.3) stellt den grundlegenden Systemaufbau exemplarisch dar.

Der IDS Server kommuniziert direkt mit allen angeschlossenen Sensoren. Für die Sensoren wird dazu das Design Pattern *Fassade* [Gamma et al., 1995, S.185] verwendet. Das bedeutet, dass Sensoren ein einheitliches Interface zur Kommunikation mit dem Server zur Verfügung

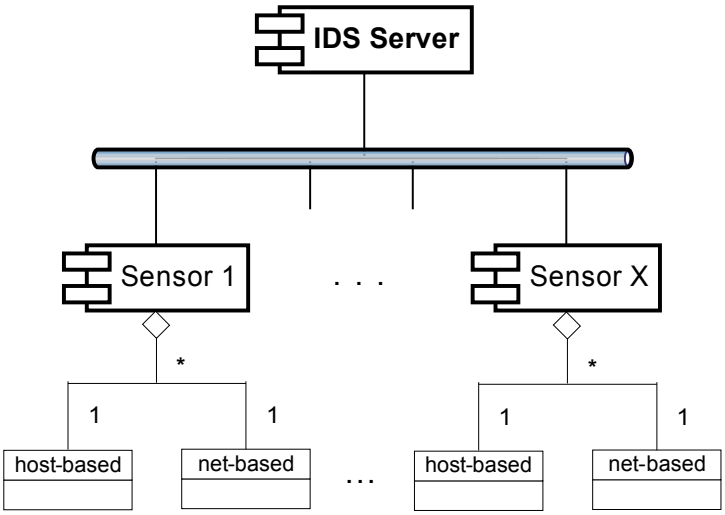


Abbildung 3.3: Systemaufbau

stellen, das unabhängig von den tatsächlich vorhandenen Host- und Netz-basierten Intrusion Detection Systemen ist. Diese widerum leiten ihre Ereignisse ungefiltert und im Rohformat an den Sensor, wo sie ggf. an das verwendete Protokoll angepasst werden. Durch diese zusätzliche Abstraktionsebene ist es möglich, nahezu beliebige Informationsquellen an einen Sensor anzuschließen, ohne wesentliche Änderungen am Server vornehmen zu müssen. Einzig eine weitere Anweisungsregel zur korrekten Extraktion der relevanten Teilinformationen ist serverseitig dafür nötig (siehe Abschnitt 4.3). Die Korrelation (siehe Abschnitt 4.4) dagegen bedarf keinerlei Änderung.

Wird in bestimmten Szenarien auf den Einsatz eines expliziten Sensors verzichtet, so kann durch Umsetzung des Design Patterns *Composite* [Gamma et al., 1995, S.163] eine direkte Kommunikation von Informationsquellen mit dem Server ermöglicht werden. Hierbei fordert das Entwurfsmuster, dass sich diese individuellen Quellen genauso verhalten wie der aus mehreren Teilen zusammengesetzte Sensor. Konkret angewandt bedeutet das jedoch nur die Implementierung desselben Kommunikationsprotokolls (siehe Abschnitt 4.2) auf den entsprechenden Hosts.

3.2.1 Sensoren

Die Sensoren sammeln alle relevanten Informationen ihres Netzsegments. Dazu zählen wie in Abschnitt 2.1 erläutert Netz-basierte Ereignisse ebenso wie Host-basierte Informationen von vorhandenen Servern oder Firewalls. Diese Ereignisse werden direkt an den IDS Server weitergeleitet. Eine lokale Speicherung zu Backup- und *drill down*-Zwecken (forensische Analysen) ist vorgesehen.

Je nach vorhandenen Quellen und lokalem Einsatzgebiet können sich Sensoren voneinander unterscheiden und demnach auch unterschiedliche Ereignisse erzeugen. Um den Wartungsaufwand so gering wie möglich zu halten, wird hier auf eine Kategorisierung und Priorisierung der Events verzichtet. Alle interessanten Events werden im entsprechenden Rohformat direkt weitergeleitet und erst auf dem IDS Server weiterführend behandelt. Dies ermöglicht auch einen umwegslosen Anschluss von Informationsquellen, falls in bestimmten Bereichen

auf einen expliziten Sensor verzichtet wird (siehe unten).

Damit die übertragene Datenmenge nicht zu groß wird, findet zudem ein Filtern auf interessante Informationen bzw. eine Beschränkung auf bestimmte Kerngebiete oder *Emergency Levels* statt. Alle anderen Ereignisse werden in lokalen Logs für einen definierten Zeitraum vorgehalten.

Diese Parameter variieren von Sensor zu Sensor und sollten an die jeweiligen Gegebenheiten angepasst werden. Wie weiter oben schon angeführt, kann es in manchen Situationen Sinn machen, von einer Bereitstellung eines dedizierten Sensors abzusehen. Oft genügt es, entsprechende Ereignisse direkt an den IDS Server weiterzuleiten. Außerdem kann die Funktionalität eines Sensor auch lokal auf einer Firewall oder einem Datenserver implementiert werden, um unnötige Maschinen und zusätzlichen Netzverkehr zu vermeiden. Diese strukturellen Alternativen sollten je nach Größe des Netzes und der angebotenen Dienste sowie dem durchschnittlichen Verkehrsaufkommen differenziert festgelegt werden.

3.2.2 Server

Der IDS Server (siehe Abbildung 3.4) verarbeit zentral alle eingehenden Ereignisse. Je nach gegebenem Einsatzgebiet sollte dieses System auf einer eigenen, leistungsfähigen Maschine betrieben werden, da gerade in großen Netzen mit viel Datendurchsatz und einer Vielzahl an Host-basierten Informationen die Filterung und Korrelation relativ aufwendig werden kann. Außerdem muss ein Leistungspuffer mit eingeplant werden, damit die Funktionaliät des IDS auch im Falle eines Angriffes (schlimmstenfalls bei *Distributed Denial of Service (DDoS)* Attacken) aufrecht erhalten werden kann.

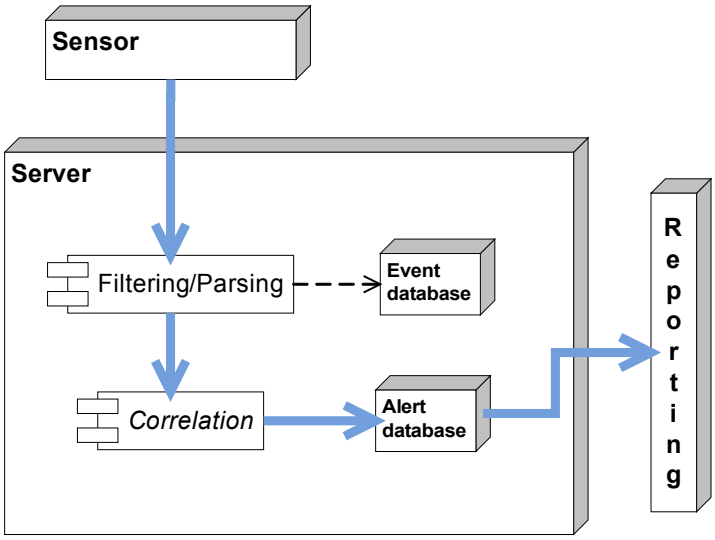


Abbildung 3.4: IDS Server Sub-Komponenten

Nach außen hin fungiert der IDS Server einerseits als Schnittstelle zum Empfangen von Sensor-Meldungen. Diese werden über eine erste Filterungsinstanz aus Backup-Gründen in die **Event-Datenbank** kopiert. Diese erste Stufe filtert die Nachrichten nach dem Sensor

ein weiteres Mal, um nur die wirklich nutzbaren Ereignisse für die Korrelation zu liefern. Die so erhaltenen tatsächlich wichtigen Events werden in Kategorien eingeteilt, mit einer Priorität versehen und in einem eigens dafür definierten Nachrichtenformat an eine zweite Korrelations-Stufe weitergereicht. Dort findet die eigentliche Korrelation statt, die erkannte Angriffe in die **Alert-Datenbank** schreibt (siehe Abschnitt 4.4).

Auf der anderen Seite bietet der IDS Server aber noch eine weitere Schnittstelle: das *User Interface*. Hier stellt ein entsprechend konzipiertes Web-Interface den Zugang zur Alert-Datenbank über eine graphische Oberfläche her. Eine detaillierte Beschreibung der vorhandenen Funktionen findet sich in Abschnitt 4.5.

Abbildung 3.4 verdeutlicht das Zusammenspiel der einzelnen Sub-Komponenten des Servers, in Abschnitt 4.2 ff. findet sich eine genaue Spezifikation der Implementierung.

4 Implementierung

Dieses Kapitel beschreibt anhand eines fortlaufenden Beispiels detailliert die Implementierung des IDS. Zunächst werden jedoch die eingesetzten Tools kurz vorgestellt.

4.1 Eingesetzte Tools

Dieser Abschnitt stellt einen kurzen Überblick über die verwendeten Tools dar. Für die Implementierung wurde keine proprietäre Software verwendet, es basiert komplett auf *open source*.

Grundsätzlich kann das IDS allein mit den Basis-Tools des folgenden Unterabschnitts rudimentär betrieben werden. Dennoch ist es essentiell, diese strukturell erforderlichen Tools durch zusätzliche Informationsquellen zu erweitern, um möglichst viele Ereignisse aus verschiedenen Bereichen mit in die Korrelation einfliessen lassen zu können. Unterabschnitt 4.1.2 liefert dazu einige ausgewählte Beispiele.

4.1.1 Basis-Tools

Diese Tools sind wesentlicher struktureller Bestandteil des IDS und daher im Gegensatz zu den darauf folgendenen Tools unverzichtbar.

syslog-ng

Auf Unix-basierten Betriebssystemen ist meist *syslogd* das Standard-Tool zur Behandlung lokaler Ereignisse. Für diese Arbeit ist es allerdings aus verschiedenen Gründen nicht ausreichend und wurde daher durch *syslog-ng* ersetzt.

Beide Tools basieren (wie auch die gesamte interne Kommunikation des IDS) auf dem standardisierten syslog-Protokoll. syslog-ng bietet darüber hinaus aber zusätzliche Möglichkeiten wie Filtern von Meldungen (mittels *Pattern Matching*) sowie einfachere Handhabung der Logquellen und Loghosts. Außerdem existiert eine Portierung für Windows-Plattformen, wodurch das Einsatzgebiet des IDS deutlich erhöht werden kann.

Eine proprietäre Version bietet zudem eine Verschlüsselung der Kommunikation zwischen Logclient und Loghost, die im Rahmen dieser Arbeit allerdings nicht notwendig ist. Prinzipiell wird davon ausgegangen, dass verschiedene Netz-Segmente bereits sicher miteinander kommunizieren, beispielsweise durch ein *VPN*. Ist dies bei einer konkreten Verwendung nicht der Fall, so könnte ein ähnlicher Sicherheitsgrad durch eine *stunnel*-basierte Lösung (`http://www.stunnel.org`) erreicht werden.

Ebenfalls wichtig zu erwähnen ist die Abwärtskompatibilität. Es ist problemlos möglich, andere auf dem syslog-Protokoll basierende Tools in die Kommunikation mit einzubinden. Effektiv getestet und verifiziert wurde die Interaktion mit *syslogd*.

SEC

SEC steht für *„simple event correlator“*) (http://kodu.neti.ee/~risto/sec/) und ist ein einfaches PERL-basiertes Tool zur Korrelation von Events. Mittels regulären Ausdrücken können beliebige eingehende Events (*Text Strings*) erkannt werden, die dann Folgeaktionen auslösen. Zudem bietet SEC eine Statusverwaltung, die es erlaubt, abhängig von einem internen Zustand unterschiedliche Aktionen bei Eintreffen eines bestimmten Events auszuführen.

Das eigentliche Erzeugen von Alerts, mit unterschiedliche Priorisierungen und Folgeaktionen sowie das Schreiben und Aktualisieren in eine Datenbank kann SEC naturgemäß aber nicht leisten. Dafür wurde ein PERL Modul implementiert, auf dessen Funktionen aus SEC heraus zurückgegriffen werden kann.

Die internen Zustände (*Contexts*) sind ein mächtiges Werkzeug, um eine generische Implementierung zu realisieren. Ein einzelnes Ereignis in einem Firewall-Log beispielsweise kann eine Vielzahl von Angriffen einleiten. Denkbar wäre ein Portscan (horizontal oder vertikal) genauso wie ein beginnender DDoS-Angriff. Es ist jedoch ebenso möglich, dass ein Paket einfach falsch geroutet wurde und keine Bedrohung gegeben ist. Um eine automatisierte Behandlung aller Möglichkeiten sicherzustellen, werden mehrere interne Zustände erzeugt, die widerum bewirken, dass entsprechende Folgeregeln auf weitere passende Events warten. Treffen diese innerhalb eines definierten Zeitraumes nicht ein, so wird der jeweilige Status gelöscht, andernfalls um eine weitere Periode verlängert. Genauere Informationen zum Vorgehen bei der Korrelation finden sich in Abschnitt 4.4.

restartd

Wie in Abschnitt 2.1 erwähnt, ist die Schwachstelle bei Host-basierter Einbruchserkennung die hohe Anfälligkeit. Um das gezielte Deaktivieren des lokalen IDS einzuschränken, wird *restartd* (http://packages.qa.debian.org/r/restartd.html) verwendet. Dieses relativ einfache Tool, das Teil des Debian Projekts ist, überprüft regelmäßig, ob ausgewählte Dienste und Programme noch ausgeführt werden. Ist dies nicht der Fall, werden sie neu gestartet und ein Log-Eintrag erstellt.

Gerade dieser Log-Eintrag kann bei der Korrelation ein entscheidender Hinweis auf einen tatsächlich erfolgten Angriff sein und muss daher in eine Event-Behandlung mit einfliessen. Generell ist es sinnvoll, dieses Tool für wesentliche und kritische Dienste auf Servern einzusetzen, da nicht nur die Ausfall-Information als Event an das IDS weitergeleitet wird, sondern der Service gleichzeitig auch wieder neu gestartet wird.

Auf weitere *proaktive* oder *reaktive* Komponenten des IDS wird in Kapitel 6 ein kurzer Ausblick gegeben.

4.1.2 Zusätzliche Informationsgewinnung

Zusätzlich zu lokalen Log- und Kernel-Meldungen muss das IDS um weitere Netz-basierte Komponenten ergänzt werden. Diese Tools sind für den Betrieb des IDS jedoch nicht zwingend notwendig. Vielmehr ist eine generische Schnittstelle vorhanden, um beliebige zusätzliche Komponenten anzubinden. Entsprechend kategorisiert können deren Ereignisse direkt in die Korrelation mit einfliessen können (siehe Abschnitt 4.3 und Abschnitt 4.4). Die folgende Auswahl stellt einige interessante Tools vor.

Snort

Snort bezeichnet sich selbst als der *de facto Standard für (Netz-basierte) Intrusion Detection* (http://www.snort.org). Diesem Anspruch wird Snort auch in der Tat gerecht. Es bietet umfassende Einbruchserkennung auf Basis von Signatur-, Protokoll und Anomalie-Analyse. Aufgrund der Komplexität ist aber eine intensive Pflege der Regeln nötig, die für sich alleine genommen schon ein eigenständiges Projekt ausfüllen könnte.

Im Fokus dieses IDP liegt daher nur eine prototypische Integration der Signatur-basierten Einbruchserkennung von Snort.

arpwatch

Arpwatch (ftp://ftp.ee.lbl.gov/arpwatch.tar.gz) verwaltet eine Liste von IP und MAC Adress-Paaren. Taucht eine neue MAC-Adresse auf oder ändert sich eine bereits bekannte Zuordnung zu einer IP-Adresse, so wird eine entsprechende Log-Meldung generiert.

Diese Art von Information lässt für sich alleine noch nicht auf einen Angriff schließen, da gerade in größeren Netzen regelmäßig Änderungen am Hardware-Pool stattfinden. Vielmehr kann sie als Initial-Event zur Erkennung eines Angriffsvektors dienen. Die entsprechende IP wird für einen vorgegebenen Zeitraum als „verdächtig“ markiert und weiter beobachtet.

pads & p0f

Pads (http://passive.sourceforge.net/) ist ein *„passive asset detection system“*. Damit können aktive Netz-Services erkannt werden. Konkret eingesetzt wird dieses Tool im Rahmen des IDPs nicht, allerdings gibt es einen interessanten Blick auf mögliche Erweiterungen. Folgt auf eine verdächtige Handlung beispielsweise das Erscheinen eines neuen Services wie die Kommunikation auf einem ungewöhnlichen Port, so sollte ein Alert erzeugt bzw. dessen Priorität erhöht werden. Auch die bloße Benutzung eines unsicheren Dienstes wie *Telnet* könnte zu einem Alarm führen.

In diesem Zusammenhang lässt sich auch *p0f* (http://lcamtuf.coredump.cx/p0f.shtml) anführen, das eine Liste von IP-Adressen mit zugehörigen Betriebssystemen pflegt, die über bekannte Paket-Signaturen erkannt werden. Sobald hier eine Änderung auftritt, sollte die entsprechende IP näher beobachtet werden.

4.2 Informationsfluss

Abbildung 4.1 stellt aufbauend auf Abbildung 3.4 eine genaue Spezifikation der Implementierung dar. Der Fokus dieses Abschnitts liegt auf der Beschreibung der Interaktion einzelner Komponenten.

Auf den sicherheitskritischen Systemen (*Main Hosts*) sind je nach vorhandenen Gegebenheiten verschiedene Komponenten zur Host-basierten Einbruchserkennung implementiert. Diese sind so konfiguriert, dass sie ihre Meldungen an den lokalen syslog-ng Dienst weiterleiten. Zusammen mit den allgemein erzeugten Log-Daten ergibt sich dadurch eine Menge an Informationen, die alle im selben Format vorliegen. Dieses basiert auf dem *syslog-Protokoll*, das in [Lonvick, 2001] spezifiziert ist. Zumal die meisten Tools dieses Protokoll schon implementieren und ihre Informationen direkt an eine entsprechende *Log Facility* weiterleiten

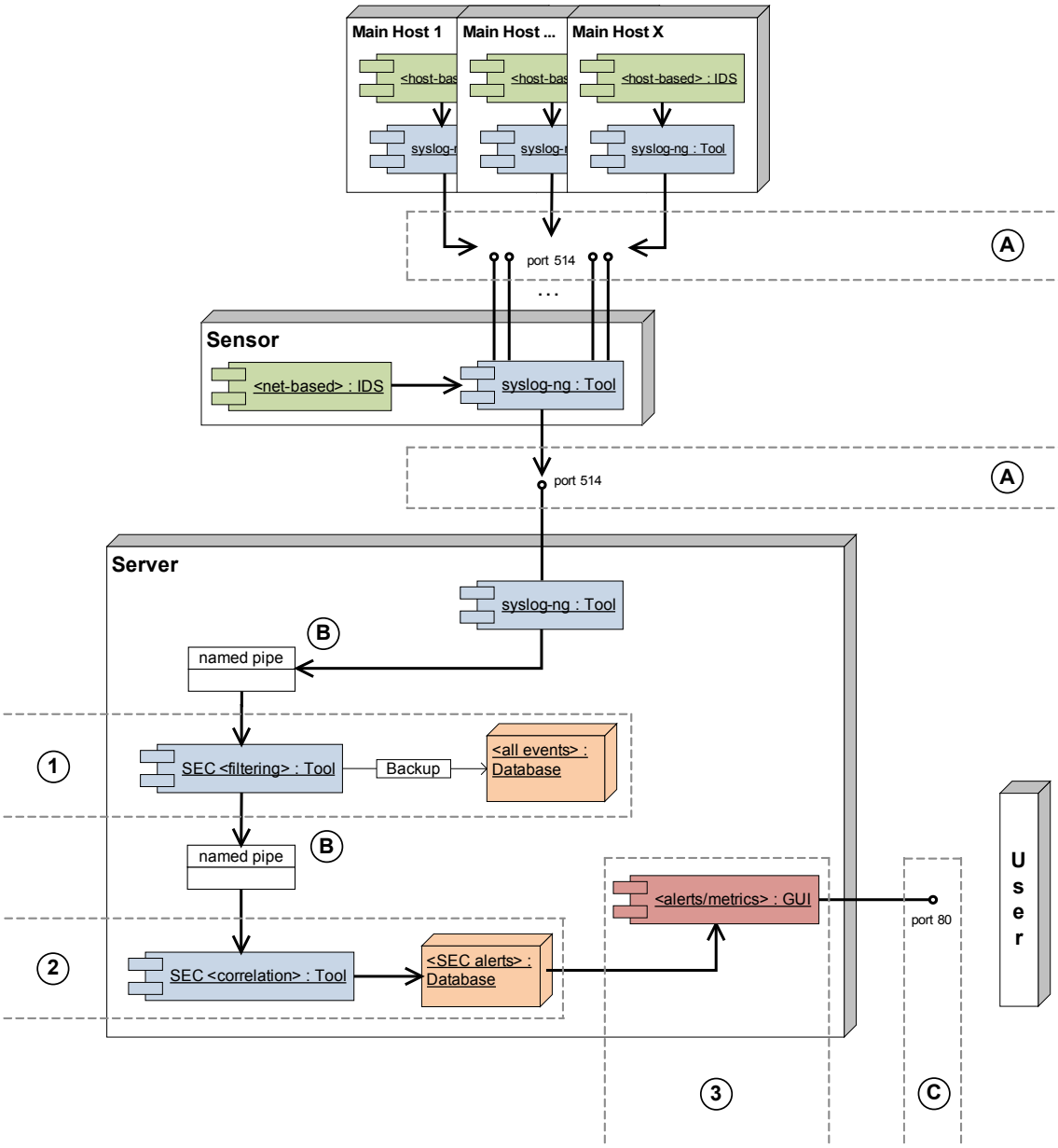


Abbildung 4.1: IDS Server Implementierung

können, ist der Anschluss von Informationsquellen relativ unaufwendig und leicht durchzuführen. Auch die Netz-basierten IDS-Komponenten auf dem Sensor werden auf diese Weise konfiguriert und dort in denselben (nachfolgend beschriebenen) Informationsfluss eingegliedert.

(A) kennzeichnet die notwendige Schnittstelle der einzelnen Komponenten. Jeder Sensor stellt einen vorläufigen *Log Host* dar, der auf einem TCP/IP-Socket Meldungen im syslog-Format erwartet. Dort folgt ein erstes Filtern unnötiger Informationen, um den Netzverkehr möglichst gering zu halten. Dabei kann sowohl nach bestimmten Log Facilities (wie etwa `AUTH`) als auch nach konkreten Tools gefiltert werden. Tabelle 4.1 stellt einen Auszug einer möglichen Konfiguration dar. Weitere Informationen hierzu finden sich in der man-Page zu syslog-ng (z.B. unter http://linux.die.net/man/5/syslog-ng.conf).

```
...
source s_all {
   internal();
   unix-stream("\dev\log");
   tcp("<IP of IDS Sensor>" port(514) max-connections(25));
};

destination d_IDS_Server {
   tcp("<IP of IDS Server>" port(514));
};

filter f_IDS {
   facility(auth,authpriv)
     or level(err..emerg)
     or match("kernel") or match("ipmon") or match("arpwatch");
};

log {
   source(s_all);
   filter(f_IDS);
   destination(d_IDS_Server);
};
...
```

Tabelle 4.1: `/etc/syslog-ng/syslog-ng.conf` – Filterung auf dem Sensor

Die reduzierte Menge aller Meldungen wird vom Sensor analog zum obigen Vorgehen an den Server übertragen. Dieser ist der endgültige Log Host, der alle eintreffenden Informationen über eine einfache syslog-ng Anweisung in eine *Named Pipe* im lokalen Dateisystem schreibt. Für die erste SEC-Instanz (siehe Abschnitt 4.3) ist dieser *FIFO*-Stapel als direkte Eingabemethode konfiguriert. Die hieraus entstehenden Ereignisse werden nach demselben Prinzip in eine weitere Pipe geleitet, die als Input für die Korrelations-Instanz von SEC dient (siehe (B) in Abbildung 4.1). Damit wird auf ganzer Linie eine *push*-Struktur umgesetzt:

an keiner Stelle muss sich eine Komponente Informationen selbst beschaffen – sie werden vielmehr vom jeweils vorherigen Subsystem in Echtzeit „weitergeschoben“.

Beispiel

Wie zu Beginn dieses Kapitels erwähnt, beziehen sich alle folgenden Code Listings der Übersichtlichkeit wegen auf ein konkretes Beispiel. Daher wird in einem fiktiven Angriffsszenario angenommen, dass ein Angreifer auf mehreren sicherheitsrelevanten Systemen versucht, sich per *ssh* Zugang zu verschaffen. Dazu testet er Logins mehrerer Standard-Benutzernamen mit schwachen Passwörtern aus, die letztendlich auf einem der Systeme zu einem erfolgreichen Zugang führen. Über die Ausnutzung einer nicht gepatchten Sicherheitslücke erlangt er root-Rechte und deaktiviert sofort den Log-Dienst.

Im Laufe dieses Kapitels wird dieses Beispiel zunächst zu Initial-Events für erste fehlgeschlagene Login-Versuche führen, die mit zunehmender Zahl Alarme steigender Priorität erzeugen. Das Deaktivieren von syslog-ng stellt schließlich einen Angriffsvektor dar, der durch die Korrelation von Ereignissen aus verschiedenen Kategorien zum höchstmöglich priorisierten Alarm führt. Nachfolgendes Diagramm veranschaulicht das Szenario schematisch.

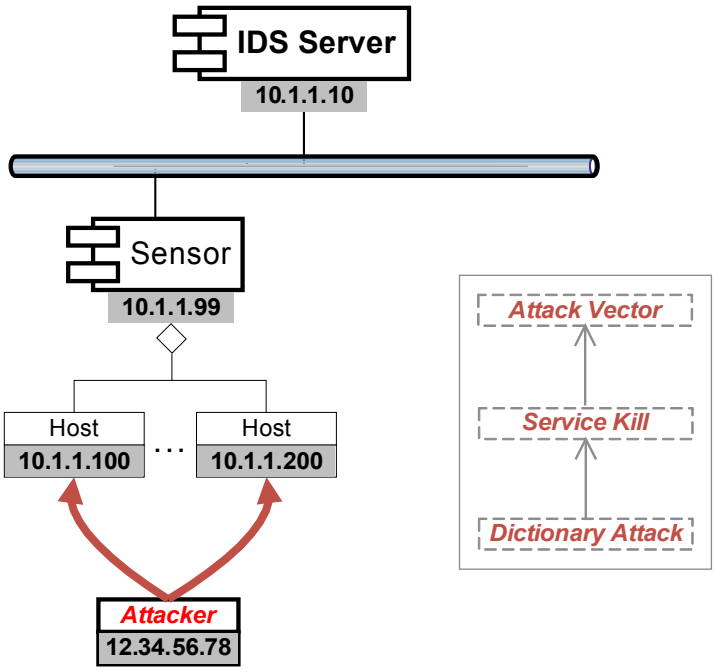


Abbildung 4.2: Beispiel-Szenario

Für den Informationsfluss ergibt sich folgender Ablauf: jeder fehlgeschlagene Login führt zu einem Log-Eintrag der folgenden Form:

```
Dec 1 20:00:00 10.0.0.100 sshd[9000]: Invalid user admin from 12.34.56.78
Dec 1 20:00:00 10.0.0.100 sshd[9000]: (pam_unix) check pass; user unknown
Dec 1 20:00:00 10.0.0.100 sshd[9000]: (pam_unix) authentication failure; (...)
```

Diese entstehen lokal auf den sicherheitskritischen Systemen, werden direkt weitergeleitet an den Sensor und dort auf ein einzelnes Ereignis reduziert:

```
Dec 1 20:00:00 10.0.0.100 sshd[9000]: Invalid user admin from 12.34.56.78
```

Anschließend wird diese Meldung an den IDS Server weitergeleitet. Kurze Zeit später folgt der Hinweis auf die Beendigung von syslog-ng:

```
Dec 1 20:01:30 10.0.0.100 syslog-ng[7500]: SIGTERM received, terminating;
```

Der nächste Abschnitt erläutert das Extrahieren der relevanten Informationen und das Erzeugen von Events für die Korrelation.

4.3 Filterung und Parsing

Eine erste SEC-Instanz wird eingesetzt, um *Events* für die Korrelation zu erzeugen. Dazu werden die eingehenden Log-Meldungen nochmals gefiltert, um nur die tatsächlich wichtigen Ereignisse weiterzuleiten. Außerdem werden dabei die relevanten Teilinformationen in einem eigenen Nachrichtenformat kategorisiert zusammengefasst. Dieser Abschnitt bezieht sich auf (1) in Abbildung 4.1.

4.3.1 Event Format

Nachfolgende *Backus-Naur-Form* (Tabelle 4.2) definiert das verwendete Nachrichtenformat. Die relevanten Teilinformationen werden aus den ursprünglichen Log-Meldungen durch *Pattern Matching* extrahiert und als *String* in die entsprechende Form gebracht. Das verwendete Trennzeichen ist ein simples Komma, da dieses – wenn überhaupt – nur im letzten Feld (*original message*) unbeabsichtigt vorkommen kann. Bei einer späteren Abarbeitung des Strings sind an dieser Stelle schon alle vorangehenden Felder ausgewertet, daher kann es hier zu keinen Konflikten mehr kommen. Sind Teilinformationen für bestimmte Ereignisse nicht verfügbar oder nicht relevant, so wird das entsprechende Feld mit `NONE` aufgefüllt.

Die so erzeugten Event-Strings werden wie in Abschnitt 4.2 beschrieben über eine Named Pipe an die zweite SEC-Instanz zur Korrelation weitergeleitet (siehe Abschnitt 4.4). Zunächst wird im nächsten Unterabschnitt aber auf die in Tabelle 4.2 angeführten *Labels* näher eingegangen.

4.3.2 Kategorisierung

Die Einteilung der Events in Kategorien ermöglicht generische Korrelationsregeln, für die die genaue Art und Herkunft der Ereignisse unerheblich ist. Jede Kategorie fasst dabei unterschiedliche Events zusammen, die aber semantisch einen ähnlichen Informationsgehalt aufweisen. Tabelle 4.3 gibt einen Überblick.

Da `INFO`-Events beispielsweise nur verdächtige Informationen über einen möglichen Angreifer darstellen, werden hier die Felder Ziel-IP und Ziel-Port auf `NONE` gesetzt. Hinweise auf Kompromittierung bestimmter Zielrechner – wie etwa das Deaktivieren eines lokalen Dienstes – dagegen verfügen über keine Quell-IP- und Quell-Port-Informationen. Das führt zu zweierlei Problemen: zum Einen muss in der Korrelation für jedes Event klar sein, über welche relevanten Teilinformationen es verfügt bzw. nach welchen korreliert wird. Dies wird

```
<message>       ::=  <label>, <priority>, <sensor>, <generator>, <src
                     IP>, <src port>, <dst IP>, <dst port>, <event ID>,
                     ORIGINAL_MESSAGE

<label>         ::=  INFO | RECON | ACCESS | COMPROM
<priority>      ::=  1..5
<sensor>        ::=  IP_ADDRESS | NONE
<generator>     ::=  SOURCE TOOL
<src IP>        ::=  IP_ADDRESS | NONE
<src port>      ::=  1..65535 | NONE
<dst IP>        ::=  IP_ADDRESS | NONE
<dst port>      ::=  1..65535 | NONE
<event ID>      ::=  00000000000..99999999999
```

Tabelle 4.2: Event Format

Label	NONE-**Fields**	**Event sources**
INFO	<dst IP>, <dst port>	pads, p0f, arpwatch
RECON		ipmon (Portscans), snort
ACCESS		telnet, sshd, login, snort
COMPROM	<src IP>, <src port>	kernel, service shutdown, snort

Tabelle 4.3: Kategorisierung

aber gerade durch die Einteilung der Ereignisse in die oben genannten Kategorien erreicht. Dadurch lassen sich auch beliebige weitere Informationsquellen an die Sensoren anschließen, sofern sie nur in die richtige Kategorie eingeordnet werden. Das weitaus schwerwiegendere Problem ist jedoch die Erkennung von Folge-Events. Wird beispielsweise einer IP eine neue MAC-Adresse zugeordnet (erkannt durch arpwatch), so kann dies ein Hinweis auf einen eingedrungenen Angreifer sein. Diese Art von Event kann allerdings nicht mit einem lokalen `COMPROM`-Ereignis eines Host-basierten IDS korreliert werden, da keine gemeinsamen Teilinformationen vorhanden sind (weder IPs noch Ports würden übereinstimmen). Um derartige Einbrüche dennoch möglichst lückenlos aufzudecken und False Positives bzw. False Negatives zu vermeiden, muss bei der Erkennung von Angriffsvektoren zusätzlich nach Vektoren von „Metainformationen“ (z.B. `INFO -> ACCESS -> COMPROM`) korreliert werden. Abschnitt 4.4 wird dieses Vorgehen beispielhaft erläutern.

4.3.3 Backup mittels 'backup.pm'

Das Systemdesign sieht an dieser Stelle des IDS auch ein Backup aller übermittelten Ereignisse vor. Dazu dient das PERL-Modul `backup.pm`, das die Funktion `putEvent(msg)` zur Verfügung stellt. Dadurch kann die Original-Nachricht bzw. der neu erzeugte Event-String in einer Datenbank abgelegt werden. Für spätere Analysen und zu Debug-Zwecken ist dies sinnvoll, allerdings wurde im Rahmen der Prototyp-Entwicklung auf eine explizite

Implementierung dieser Funktion verzichtet. Das notwendige Rahmenwerk dazu ist jedoch vorhanden.

Um ein eigenes PERL-Modul einzubinden, muss SEC mit der Option *-intevents* gestartet werden, die ein `SEC_STARTUP`-Event erzeugt. Eine entsprechende Initialisierungs-Regel kann dann bei Eintreffen dieses Events die Aktion `eval %a ( require 'backup.pm' )` ausführen, wodurch alle Modul-Methoden verfügbar werden. Ein analoges Vorgehen ist auch in Unterabschnitt 4.4.4 zur Einbindung des Korrelations-Moduls nötig.

4.3.4 Beispiel-Regel

Der folgende Auszug aus der SEC-Konfigurationsdatei stellt die resultierenden Arbeitsschritte dieses Abschnitts exemplarisch an dem in Abschnitt 4.2 eingeführten Beispiel vor. Dazu muss zunächst das folgende sshd-Event erkannt und in das neue Nachrichtenformat umgewandelt werden. Die relevanten Teilinformationen sind hervorgehoben.

```
Dec 1 20:00:00 10.0.0.100 sshd[9000]: Invalid user admin from 12.34.56.78
```

Die entsprechende Regel dazu findet sich in nachfolgender Tabelle 4.4.

```
1 desc=sshd 'Invalid User' events
2 type=Single
3 ptype=RegExp
4 pattern=\S+\s+\S+\s+\S+\s+(\S+) (\S+)\[\S+\]: Invalid user \S+ from (\S+)
5 action=eval %a ( putEvent("$0") ); \
          eval %b ( $id="3" . `date +%S%N`; chop($id); chop($id); $id ); \
          write pipe ACCESS, 1, $1, $2, $3, NONE, $1, NONE, %b, $0;
```

Tabelle 4.4: `Beispiel-Filter (1)`

1. `desc:` „Description“ – dient nur zur Beschreibung der nachfolgenden Regel und wird von SEC ignoriert.
2. `type:` Legt den Typ der Regel fest. Im folgenden wird nur `Single` benutzt, also eine einzige unabhängige Regel. Andere Typen wären `Pair` oder `PairWithWindow`.
3. `ptype:` Legt die Mächtigkeit des Pattern Matchings fest, in dieser Arbeit immer `RegExp` (PERL Regular Expressions, http://perldoc.perl.org/perlre.html).
4. Nach dieser Initialisierung beginnt die eigentliche Abarbeitung des Event-Strings. Passt dieser auf den regulären Ausdruck, so werden die nachfolgend definierten `actions` ausgeführt, andernfalls wird der String an die nächste Regel weitergereicht. Passt er auf keine Regel, so erfolgt keine Behandlung.
   Die PERL-basierten regulären Ausdrücke ermöglichen das Extrahieren von Teilinformationen. Alle geklammerten Ausdrücke können innerhalb der `action`- Statements über $1, $2, ... usw. referenziert werden, $0 enthält den gesamten String.

In dieser Regel stellt die erste Klammer `(\S+)` eine Folge von nicht-leeren Zeichen dar, die die Quell-IP der Ursprungsnachricht beinhaltet. Die nächste Klammer erfasst den Namen des Tools, das die Log-Meldung erzeugt hat, und die dritte Klammer extrahiert die IP des Angreifers (siehe vorangehenden markierten Log-Eintrag). Quell- und Ziel-Port sind in diesem Event nicht vorhanden, dementsprechend werden sie später auf `NONE` gesetzt.

(5) Falls der reguläre Audruck auf den Event-String passt, werden alle `action`-Statements abgearbeitet. Für die Filter-Regeln sind diese immer dreigeteilt:

- Über `eval %a (..)` lassen sich Funktionen des entsprechend eingebunden PERL Backup-Moduls aufrufen. Rückgabewerte befinden sich anschließend in der internen SEC-Variable `%a`. Mit dieser ersten Aktion wird der gesamte Nachrichten-String wie in Unterabschnitt 4.3.3 beschrieben aus Backup-Gründen gesichert.
- Das zweite Statement dient zur Erzeugung einer *Unique ID*, damit die Events später eindeutig referenziert werden können. Dazu wird ebenfalls über die `eval`-Funktion PERL-Code ausgeführt, der eine elfstellige Zahl aus den aktuellen Sekunden und Nanosekunden erstellt. Die ID kann danach über `%b` benutzt werden.
- Hier werden die Teilinformationen zu einem neuen Event (spezifiziert in Tabelle 4.2) zusammengesetzt und in die nachfolgende Pipe geschrieben (siehe unten).

Nach diesem Schritt entsteht folgendes neue Event, das im nächsten Abschnitt in der Korrelation wieder aufgegriffen wird.

```
"ACCESS, 1, 10.1.1.100, sshd, 12.34.56.78, NONE, 10.1.1.100, NONE, 32538457256,
      Dec 1 20:00:00 10.0.0.100 sshd [9000]: Invalid user admin from 12.34.56.78"
```

Analog ist für das Erkennen der syslog-ng-Deaktivierung folgende Regel (Tabelle 4.6) nötig,

```
desc=kill event
type=Single
ptype=RegExp
pattern=\S+\s+\S+\s+\S+\s+(\S+) (\S+)\[\S+\]: SIGTERM received
action=eval %a ( putEvent("$0") );
       eval %b ( $id="6" . `date +%S%N`; chop($id); chop($id); $id ); \
       write pipe COMPROM, 3, $1, $2, NONE, NONE, $1, NONE, %b, $0;
```

Tabelle 4.6: `Beispiel-Filter (2)`

die zu diesem Event führt:

```
"COMPROM, 3, 10.1.1.100, syslog-ng, NONE, NONE, 10.1.1.100, NONE, 65135698403,
      Dec 1 20:01:30 10.0.0.100 syslog-ng[7500]: SIGTERM received, terminating;"
```

4.4 Korrelation

Dieser Abschnitt beschreibt die Korrelation von Ereignissen. Grundlage dafür ist zunächst eine Schwellwert-Analyse von Ereignissen gleicher Kategorie. In einem weiteren Schritt folgt danach die Erkennung von Angriffsvektoren, also die Analyse von Beziehungen über Kategorie-Grenzen hinweg.

4.4.1 Überblick

Entscheidend bei der Korrelation ist eine *zustandsbasierte* Erkennung von Folge-Events. Diese Zustände werden von Initial-Ereignissen erzeugt und beinhalten deren semantische und technische Informationen. So kann bei einem nachfolgenden Ereignis anhand der intern gesetzten Zustände geprüft werden, ob bereits Events mit sich überschneidenden Teilinformationen eingetroffen sind. In diesem Fall wird der entsprechende Zustand aktualisiert und gegebenenfalls ein Alarm erzeugt. Ist bereits ein Alarm vorhanden, so wird dieser erneuert und eventuell höher priorisiert. Abbildung 4.3 zeigt das allgemeine Zustandsdiagramm.

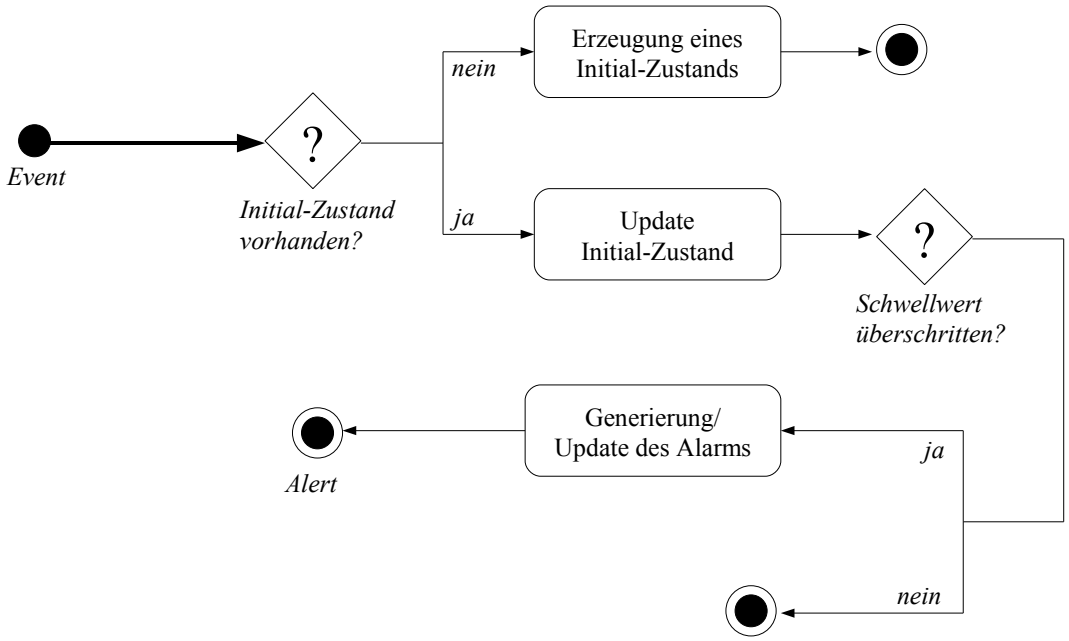


Abbildung 4.3: Zustandsdiagramm

Zur Realisierung dieser Zustandsübergänge sind vier spezielle SEC-Regeln für jede Ereignis-Kategorie notwendig. Sollen Angriffsvektoren, also Beziehungen über Kategorie-Grenzen hinweg erkannt werden, so sind pro Vektor weitere zwei Regeln nötig. Unterabschnitt 4.4.5 liefert hierfür jeweils ein Beispiel.

4.4.2 Schwellwert-Analyse

Die Schwellwert-Analyse ist das grundlegende Vorgehen zur Zusammenfassung von Ereignissen. Je nach Anzahl von korrelierenden Ereignissen der selben Kategorie wird ein entspre-

chend priorisierter Alarm erzeugt. Abbildung 4.4 stellt eine Übersicht über die verschiedenen Korrelationspfade dar.

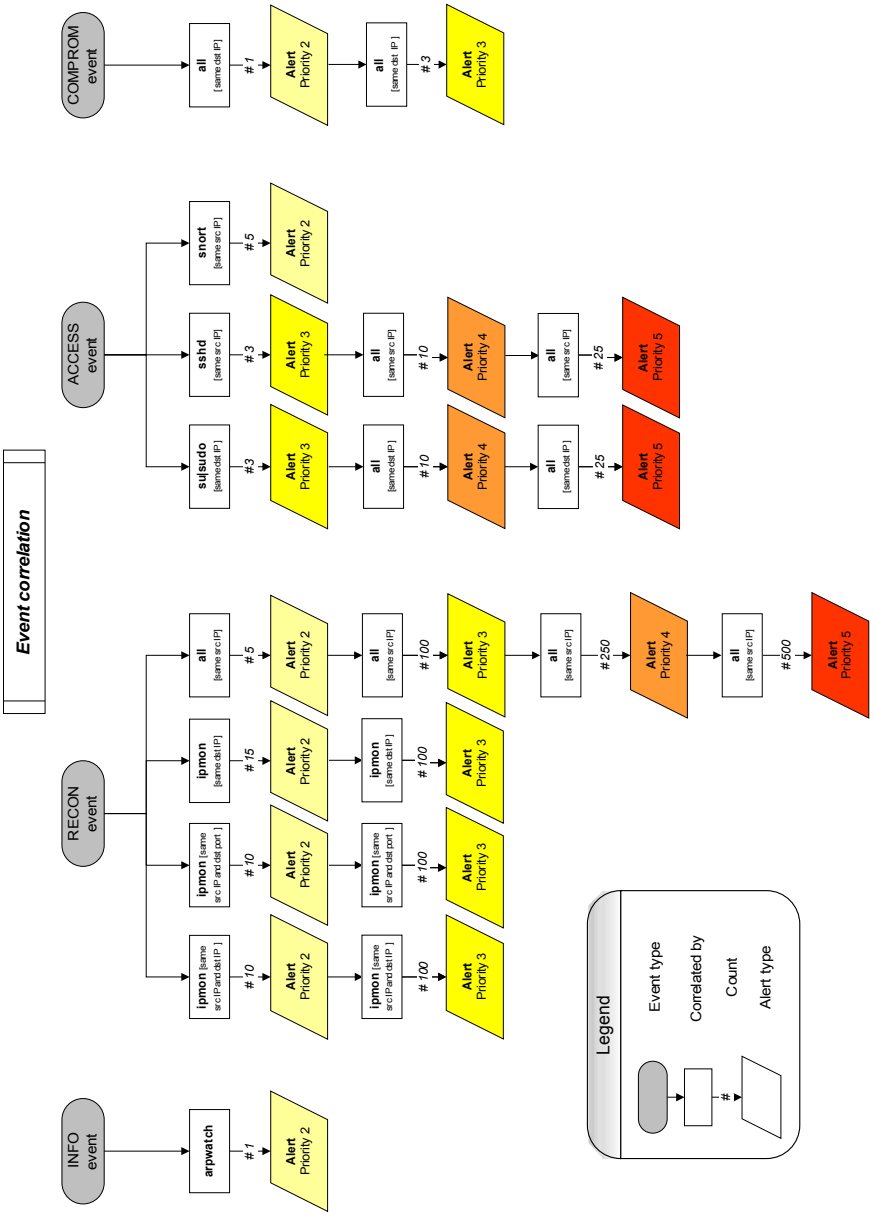


Abbildung 4.4: Korrelationspfade

Die einzelnen Schwellwerte können über einen PERL-Hash variabel mit in die Konfiguration von SEC mit aufgenommen werden. Wichtig hierbei ist eine sinnvolle Definition von Schwellwerten für verschiedene Prioritätsstufen und Kategorien.

4.4.3 Vektor-Erkennung

Zusätzlich zur Schwellwert-Erkennung ist eine Analyse von Ereignissen unterschiedlicher Kategorien sinnvoll. Diese Analyse basiert auf der Annahme, dass sich ein Angreifer zunächst verdächtig im Netz verhält (z.B. Wechsel der MAC-Adresse) oder Informationen über das Netz sammelt (z.B. über Portscans). Danach folgt der Versuch, sich Zugang zu einem System zu verschaffen, entweder über übliche Wege wie ssh oder über das Ausnutzen einer bekannten Schwachstelle. Abschließend folgen Kompromittierungs-Ereignisse auf dem entsprechenden Zielsystem. Abbildung 4.5 zeigt ein Schema der möglichen Angriffsvorgänge, das zudem ein Fehlen einzelner Zwischenschritte berücksichtigt.

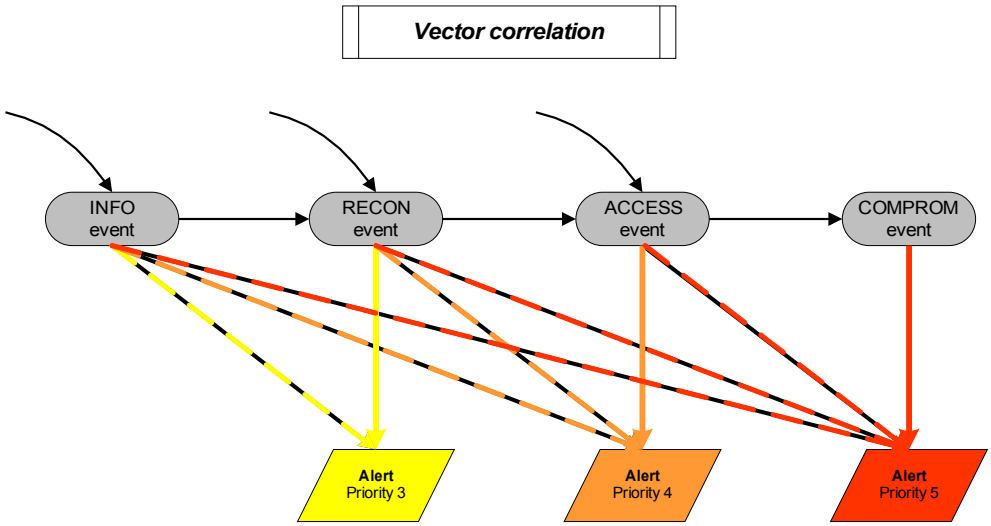


Abbildung 4.5: Angriffsvektoren

Bei dieser Analyse handelt es sich bei den Initial-Events nicht notwendigerweise um Einzel-Ereignisse. Vielmehr können die Startpunkte eines Angriffsvektors bereits aus entsprechenden Schwellwert-Alarmen bestehen. Unterabschnitt 4.4.5 stellt einen möglichen Angriffs- und Erkennungsvorgang exemplarisch dar.

4.4.4 Das PERL Modul 'alert.pm'

Analog zu Unterabschnitt 4.3.3 wird auch für die Korrelation auf ein externes PERL Modul zugegriffen. Dieses stellt Methoden zur Verfügung, durch die ein Hinzufügen und Modifizieren von Alarmen in der Alert-Datenbank möglich ist.

`putAlert()` ermöglicht das Erstellen eines Alarms. Dazu müssen alle Ereignisse, die im entsprechenden internen SEC-Zustand gespeichert wurden, extrahiert und als einzelne Ereignisse in die Datenbank mit aufgenommen werden. Anschließend folgt die Erzeugung des Alarms, der über Verweise auf die zugehörigen Ereignisse verfügt. Dadurch lässt sich ein Backtracking eines Alarms bis hin zu den originalen syslog-Meldungen realisieren.

`updateAlert()` fügt für einen bereits vorhandenen Alarm weitere Folge-Events hinzu. Wichtig bei dem Update ist das Überprüfen der Schwellwerte, um gegebenenfalls die Priorität des Alarms zu erhöhen.

4.4.5 Beispiel-Regel

Im Folgenden wird das Beispiel aus Abschnitt 4.2 wieder aufgegriffen. Zunächst muss dementsprechend ein Regelablauf zur Erkennung von mehreren fehlgeschlagenen Login-Versuchen existieren. Dieser führt nach Überschreitung des Schwellwerts zu einem Alarm, der mit Eintreffen von weiteren Ereignissen zunehmend höher priorisiert wird. Im Anschluss daran muss ein lokales Kompromittierungs-Event zum höchtmöglichen Alarm führen. Tabelle 4.7 und Tabelle 4.9 erläutern das zustandsbasierte Vorgehen.

```
① desc=creating first context
   type=Single
   ptype=RegExp
   pattern=^ACCESS, \S+, \S+, sshd, (\S+), \S+, \S+, \S+, \S+, (.*)
   context=!REMLOGIN_FROM_$1 && !COR_REMLOGIN_FROM_$1
   continue=TakeNext
   action=eval %o ( $remlogin{"$1"} = {} ); \
          eval %o ( $remlogin{"$1"}->{"time"} = %u ); \
          create REMLOGIN_FROM_$1 900 eval %o ( delete $remlogin{"$1"} )

② desc=saving events if correlating alert doesnt exist yet
   type=Single
   ptype=RegExp
   pattern=^ACCESS, \S+, \S+, sshd, (\S+), \S+, (\S+), \S+, (\S+), (.*)
   context=REMLOGIN_FROM_$1 && !COR_REMLOGIN_FROM_$1
   continue=TakeNext
   action=eval %o ( $remlogin{"$1"}->{"$3"} = "$2;;sshd;;$4" ); \
          set REMLOGIN_FROM_$1 900 eval %o ( delete $remlogin{"$1"}

③ desc=putting inital alert and new context creation
   type=Single
   ptype=RegExp
   pattern=^ACCESS, \S+, \S+, sshd, (\S+), \S+, (\S+), \S+, \S+, (.*)
   context=REMLOGIN_FROM_$1 && !COR_REMLOGIN_FROM_$1 && \
           =( scalar(keys(%{$remlogin{"$1"}})) > $thresholds{"remlogin"}->{"prio2"} )
   action=eval %x ( putAlert($remlogin{"$1"},"remote access fails", \
                    0,2,"$1","$2","22",$thresholds{"remlogin"}->{"prio2"}) ); \
          delete REMLOGIN_FROM_$1; \
          create COR_REMLOGIN_FROM_$1 900 eval %o ( delete $remlogin{"$1"} ); \
          event COR_ACCESS, 2, $2, sshd, $1, NONE, $2, 22, %x

④ desc=updating existing alert
   type=Single
   ptype=RegExp
   pattern=^ACCESS, \S+, \S+, sshd, (\S+), \S+, (\S+), \S+, (\S+), (.*)
   context=COR_REMLOGIN_FROM_$1
   action=eval %o ( updateAlert($remlogin{"$1"},"$2","sshd","$3","$4", \
                    $thresholds{"remlogin"},"yes") ); \
          set COR_REMLOGIN_FROM_$1 900 eval %o ( delete $remlogin{"$1"} ); \
          eval %x ( $remlogin{"$1"}->{"ID"} . ', ' . $remlogin{"$1"}->{"time"} ); \
          event COR_ACCESS, 2, $2, sshd, $1, NONE, $2, 22, %x
```

Tabelle 4.7: `Beispiel Schwellwert-Analyse`

Tabelle 4.7 zeigt beispielhaft an `ACCESS`-Events die vier notwendigen Regeln zur Realisie-

rung einer Schwellwert-Analyse. Dabei wird das zustandsbasierte Vorgehen aus Unterabschnitt 4.4.1 umgesetzt.

(1) Die erste Regel dient bei Eintreffen des Initial-Events zur Erzeugung eines internen Zustands (*„context“*). Die Anweisung des Pattern Matchings bildet dabei direkt das Nachrichtenformat spezifiziert in Tabelle 4.2 ab. Entscheidend für diese Regel ist, dass bisher kein `ACCESS`-Event derselben Quelle eingetroffen ist. Diese Bedingung wird durch das `context`-Statement umgesetzt.
Die resultierenden Aktionen für das Initial-Ereignis sind einerseits das Erzeugen der technischen Zustandsdaten. Dies wird durch einen PERL-Hash realisiert, der durch `"remlogin"` und `"$1` einen `ACCESS`-Zustand für die Quelle `$1` beschreibt. Zudem hält dieser Hash auch die Startzeit des Vorgangs (`%u`). Andererseits muss ein entsprechender Zustand auch intern für SEC erzeugt werden, was durch die dritte Aktionsanweisung umgesetzt wird. Dabei beträgt die Lebenszeit des Zustands 900 Sekunden; nach dessen Ablauf wird als Folgeaktion der passende PERL-Hash wieder gelöscht.
Die Anweisung `continue=TakeNext` bewirkt das anschließende Weiterreichen des Ereignisses an die nächste Regel.

(2) Durch die erste Regel wurde nun bereits ein interner Zustand erzeugt. Als nächster Schritt muss das Initial-Ereignis im PERL-Hash abgelegt werden. Dazu wird im Hash die ID des Ereignisses als neuer Schlüssel hinzugefügt. Dessen Wert ist die Originalnachricht `$4`, ergänzt um weitere Informationen wie Ziel-IP und Informationsquelle. Dadurch lassen sich erweiterte Funktionen für die GUI umsetzen; genauere Informationen hierzu finden sich im Quellcode.
Diese Regel dient zudem auch zur Speicherung von Folge-Events. Einzige Bedingung dafür ist das Vorhandensein des entsprechenden Zustandes, das durch ein Initial-Ereignis allerdings bereits sichergestellt wird.
Die zweite Aktion erneuert die Lebenszeit des Zustandes.

(3) Das Erzeugen des eigentlichen Alarms findet erst bei Überschreitung eines Schwellwertes statt. Dies wird durch das `context`-Statement der dritten Regel überprüft. Dazu wird innerhalb von `=( ... )` PERL-Code ausgeführt, der die Anzahl der Ereignisse des zugehörigen PERL-Hashes mit einem vordefinierten Schwellwert vergleicht. Ist dieser überschritten, so werden die entsprechenden Aktionen ausgeführt. Wichtig hierbei ist, dass diese Überprüfung nur stattfindet, wenn bereits ein entsprechender SEC-Zustand vorhanden ist (und andererseits aber noch kein Zustand für einen bereits erzeugten Alarm).
Zur Erzeugung des Alarms wird auf das externe PERL-Modul (siehe Unterabschnitt 4.4.4) zugegriffen und der entsprechenden Methode alle relevanten Teilinformationen übergeben. Desweiteren wird der ursprüngliche Zustand gelöscht und ein neuer (`COR_REMLOGIN_FROM_$1`) erzeugt. Dadurch wird die Abarbeitung der bisherigen drei Regeln verhindert (siehe `context`-Statements), um für die gesamte Lebenszeit des neuen Zustandes (und damit des erzeugten Alarms) nur noch die letzte Regel zu verwenden.
Entscheidend für die Erkennung von Angriffsvektoren ist die Erzeugung eines neuen Events in der letzten Anweisungszeile. Dieses ist für einen weiteren Regelsatz bestimmt, auf den im Folgenden ebenfalls genauer eingegangen wird.

④ Die letzte Regel dient zur Aktualisierung eines bereits vorhanden Alarms. Dementsprechend wird mit der `context`-Anweisung überprüft, ob bereits ein entsprechender Zustand vorhanden ist. Die folgenden Aktionen führen widerum eine externe Methode zur Datenbank-Aktualisierung aus (siehe Unterabschnitt 4.4.4). Desweiteren wird die Lebenszeit des Zustandes verlängert. Zudem wird der PERL-Hash aktualisiert und ein weiteres Event zur Vektor-Analyse erzeugt.

Wie bereits angesprochen sind zur Erkennung von Angriffsvektoren weitere zwei Regeln notwendig, die in Tabelle 4.9 dargestellt sind.

```
① desc=creating context and hash for attack vector
   type=Single
   ptype=RegExp
   pattern=^COR_ACCESS, \S+, \S+, (\S+), (\S+), \S+, (\S+), \S+, (\S+), (.*)
   context=!COR_ACCESS_ON_$3
   action=eval %o ( $vector{"$3"} = {} ); \
          eval %o ( $vector{"$3"}->{"time"} = "$5" ); \
          eval %o ( $vector{"$3"}->{"from"} = "$2" ); \
          eval %o ( $vector{"$3"}->{"$4"} = "$3;;$1;;not used" ); \
          create COR_ACCESS_ON_$3 900 eval %o ( delete $vector{"$3"} )

② desc=putting alert for vector
   type=Single
   ptype=RegExp
   pattern=^COMPROM, (\S+), \S+, (\S+), \S+, \S+, (\S+), \S+, (\S+), (.*)
   context=COR_ACCESS_ON_$3
   action=eval %o ( $comprom{"$3"} = {} ); \
          eval %o ( $comprom{"$3"}->{"time"} = %u ); \
          eval %o ( $comprom{"$3"}->{"$4"} = "$3;;$2;;$5" ); \
          eval %x ( putAlert($comprom{"$3"},"local comprom event", \
                    0,"$1","","$3","",1) ); \
          eval %o ( $vector{"$3"}->{"from"} ); \
          delete COR_REMLOGIN_FROM_%o; \
          eval %o ( @fields = split(/, /, %x); $fields[0] ); \
          eval %o ( $vector{"$3"}->{"%o"} = "$3;;$2;;not used" ); \
          eval %x ( putAlert($vector{"$3"},"attack vector detected", \
                    1,5,$vector{"$3"}->{"from"},"$3","",2) ); \
          delete COR_ACCESS_ON_$3
```

Tabelle 4.9: `Beispiel Vektor-Analyse`

Der Ablauf der Regeln zur Erkennung von Angriffsvektoren (Tabelle 4.9) gleicht grundsätzlich dem zustandsbasierten Vorgehen beschrieben in Unterabschnitt 4.4.1, allerdings unterscheidet sich die Implementierung von der Schwellwert-Analyse in Tabelle 4.7. Grundsätzlich reicht hier ein Folge-Event einer weiteren Kategorie aus, denn damit wird bereits der höchtmöglich priorisierte Alarm erzeugt und ein Angriff mit größtmöglicher Wahrscheinlichkeit erkannt.

Die angeführten Regeln stellen folgenden Vektor, bezogen auf Abbildung 4.5 dar (Abbildung 4.6):

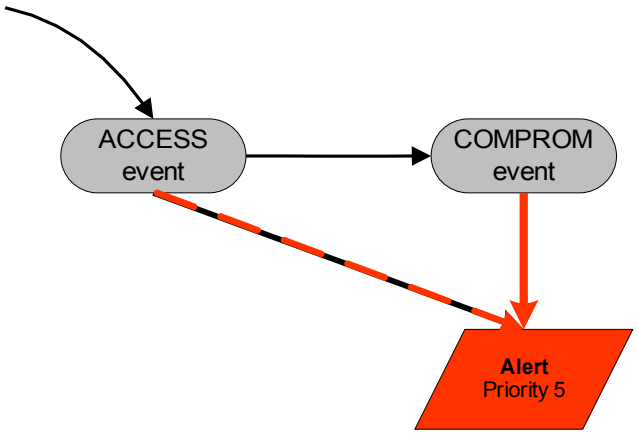


Abbildung 4.6: Beispiel Angriffsvektor

In den folgenden Punkten unterscheidet sich die Implementierung:

① In den Regeln ③ und ④ in Tabelle 4.7 wurde bereits ein neues Event erzeugt. Dieses wird mit dem Pattern Matching dieser Regel abgefangen, falls nock kein entsprechender korrelierter Zustand vorhanden ist. Durch die Aktionsliste wird ein neuer PERL-Hash erzeugt, der die ursprünglichen Informationen des vorangehenden Alarms beinhaltet. Außerdem folgt die Erzeugung des korrelierten Zustandes.

② Folgt nun ein Ereignis der `COMPROM`-Kategorie, so kann davon ausgegangen werden, dass die Login-Versuche erfolgreich waren und dementsprechend muss der bereits vorhandene Alarm entsprechend aktualisiert und höhergestuft werden. Dazu muss zunächst das `COMPROM`-Ereignis über das externe PERL-Modul in die Datenbank geschrieben werden. Außerdem kann der noch existierende Zustand aus den vorangehenden Regeln gelöscht werden.
Anschließend folgt das Erzeugen des neuen Alarms, der die beiden bisherigen beinhaltet sowie das Löschen des korrelierten Zustandes.

4.5 GUIs und Metriken

In Abschnitt 4.4 wurde detailliert das Vorgehen bei der Korrelation von Ereignissen erläutert. Die daraus entstandenen Alarme können nun über die Benutzer-Schnittstelle dargestellt werden. Dabei können einerseits genaue Informationen über den Angriffsverlauf abgerufen werden, andererseits gliedert sich der Alarm in die entsprechende Statistik der letzten beiden Wochen mit ein. Die nächsten beiden Unterabschnitte stellen die Funktionen kurz vor.

4.5.1 alertGUI

Die sogenannte `alertGUI` ist eine PERL/CGI-basierte Web-Oberfläche. Abbildung 4.7 gibt einen Überblick.

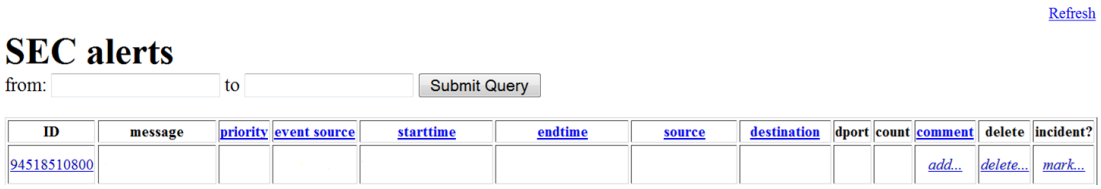


Abbildung 4.7: Alarm-Darstellung

Die Web-Oberfläche bietet die Möglichkeit, Alarme nach Priorität, Informationsquellen, Zeit, Ziel, Quelle und Kommentaren zu sortieren. Desweiteren können über die Alarm-ID alle zugehörigen Ereignisse eingesehen werden.

Wichtige Interaktionsmöglichkeiten sind das Hinzufügen von Kommentaren sowie das Löschen von Alarmen. Außerdem kann ein Zeitraum für die darzustellenden Alarme gewählt werden. Interessant für die Erzeugung von Metriken ist eine Markierungsmöglichkeit des Alarms als tatsächlichen Sicherheitsvorfall. Daraus kann dann eine Statistik über False Positives generiert werden (siehe Unterabschnitt 4.5.2).

Je nach Priorität des Alarms wird die entsprechende Zeile farblich markiert. Ein Beispiel für die Darstellung mehrerer Alarme findet sich in Abbildung 5.2.

4.5.2 metricGUI

Zur Darstellung von verschiedenen Metriken wurden PHP-basierte interaktive Diagramme implementiert (siehe http://teethgrinder.co.uk/open-flash-chart/). Diese Diagramme zeigen zum Einen eine Übersicht über alle Alarme der aktuellen und vorangehenden Woche. Dabei wird durch farbliche Balken nach Priorität unterschieden (siehe beispielsweise Abbildung 5.1). Zum Anderen kann eine Statistik über False Positives dargestellt werden.

Abbildung 4.8 gibt einen Überblick.

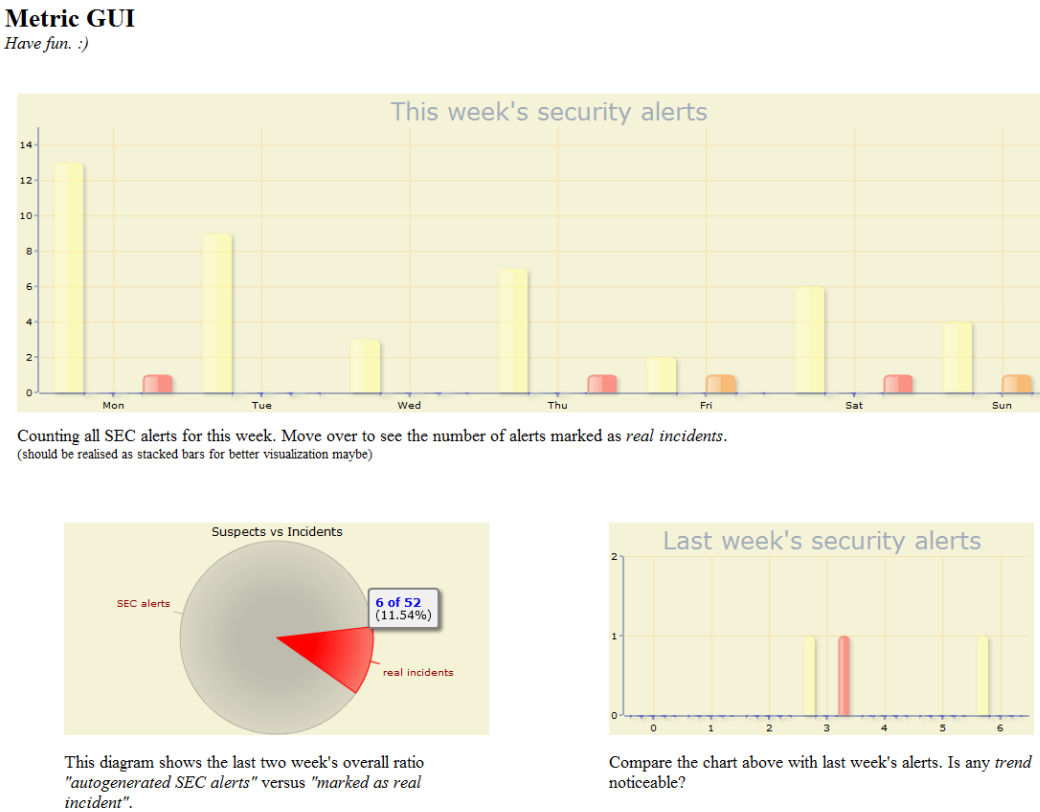


Abbildung 4.8: Metrik-Darstellung

5 Testeinsatz und Ergebnisse

In Kapitel 3 wurde die grundlegende Infrastruktur der IDS Komponenten näher erläutert. Dabei wurde deutlich, dass je nach Einsatzgebiet alternative Aufstellungen in Frage kommen. Im Rahmen von Testeinsätzen wurden zwei verschiedene Szenarien betrachtet, deren zugrunde liegende Struktur sich stark unterscheidet. Die eigentliche Funktionalität – nämlich Einbruchserkennung und metrikbasiertes Reporting – bleibt dennoch transparent erhalten.

5.1 LMU München

Zu Test- und Demonstrationszwecken wurden zwei virtuelle Maschinen aufgesetzt, die beide direkten und ungeschützten Zugang zum Internet aufweisen. Ein produktiver Einsatz mit mehreren (virtuellen) Sensoren und zusätzlichem Anschluss von vorhandenen Firewalls und Datenservern wäre denkbar, ist aber für diesen konkreten Prototyp nicht von Bedeutung. Die eigentliche Aufgabe ist vielmehr, die zugrunde liegende Implementierung zu testen und zu präsentieren.

Daher beschränkt sich das Auswerten von Events auf lokale Fehler wie Kernel-Meldungen oder Dienst-Ausfälle sowie auf fehlgeschlagene Login-Versuche. Dennoch sind damit sowohl einfache Schwellwert-Alerts als auch komplexe Angriffsvektoren abbildbar.

Überraschenderweise ergaben sich interessante, ganz reale Einblicke in Häufigkeit und Intensität von *Wörterbuchangriffen* auf das Netz des Informatik-Lehrstuhls der LMU (und damit vermutlich auch auf das ebenfalls vom LRZ betriebene Münchner Wissenschaftsnetz). Abbildung 5.1 vermittlet einen Eindruck der Bedrohungslage über einen Zeitraum von einer Woche.

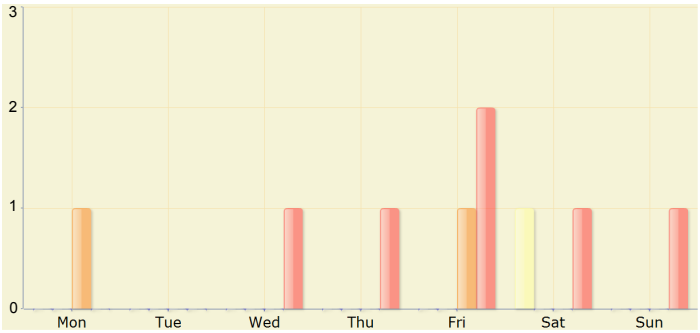


Abbildung 5.1: Häufigkeit von Wörterbuchangriffen

Die x-Achse ist aufgeteilt in 7 Tage, die y-Achse stellt die Anzahl der Wörterbuchangriffe dar. Es finden sowohl Angriffe mit einigen wenigen Versuchen (gelb) als auch stundenlange Prüfungen mit mehreren Tausend Einzel-Ereignissen statt. Aus dem Diagramm lässt sich

erkennen, dass die an das Internet angeschlossenen Komponenten fast täglich mit Einbruchsversuchen konfrontiert sind. Am Wochenende nimmt die Konzentration der Angriffe zu, da dort vermutlich mit weniger Aufmerksamkeit der Systemadministration gerechnet wird.

SEC alerts

from: 2008-11-03 00:00:01 to 2008-11-09 23:59:59 Submit Query

ID	message	priority	event source	starttime	endtime	source	destination	dport	count	comment	delete	incident?
94514460600	remote access fails	5	sshd	2008-11-09 22:02:42	2008-11-09 22:41:55	141.85.37.186	141.84.218.23 141.84.218.26	22	2274	add...	delete...	marked
92304793000	remote access fails	2	sshd	2008-11-08 19:37:15	2008-11-08 19:37:42	218.56.61.114	141.84.218.23 141.84.218.26	22	12	add...	delete...	marked
93741152500	remote access fails	5	sshd	2008-11-08 05:07:29	2008-11-08 06:02:38	67.221.35.80	141.84.218.23 141.84.218.26	22	3397	add...	delete...	marked
91049195200	remote access fails	4	sshd	2008-11-07 16:23:51	2008-11-07 16:31:17	119.70.132.135	141.84.218.23 141.84.218.26	22	104	add...	delete...	marked
91822312700	remote access fails	5	sshd	2008-11-07 12:29:13	2008-11-07 12:38:15	208.116.160.23	141.84.218.26 141.84.218.23	22	503	add...	delete...	marked
94979833800	remote access fails	5	sshd	2008-11-07 08:48:41	2008-11-07 09:24:53	203.163.253.230	141.84.218.26 141.84.218.23	22	641	add...	delete...	marked
95379845000	remote access fails	5	sshd	2008-11-06 01:30:45	2008-11-06 02:35:04	83.13.131.190	141.84.218.23 141.84.218.26	22	4106	add...	delete...	marked
91024811000	remote access fails	5	sshd	2008-11-05 15:52:54	2008-11-05 20:55:45	221.194.138.3	141.84.218.23 141.84.218.26	22	8497	add...	delete...	marked
95616399400	remote access fails	4	sshd	2008-11-03 14:50:49	2008-11-03 15:10:07	61.153.18.194	141.84.218.23 141.84.218.26	22	412	add...	delete...	marked

page: **0**

Abbildung 5.2: Erzeugte Alerts einer Woche

Die zu Abbildung 5.1 gehörenden Alerts sind in Abbildung 5.2 abgebildet. Erkennbar ist, dass sich ein Angriff nie auf eine einzelne Maschine beschränkt, sondern (zumindest in diesem Szenario) immer alle verfügbaren Systeme mit einbezieht. Aus Start- und Endzeit eines Angriffes sowie der Anzahl der Login-Versuche lässt sich folgende Tabelle 5.1 berechnen:

ID	Time	Logins	Logins/minute
95379845000	1h 04m 19s	4106	63.84
93741152500	0h 55m 09s	3397	61.60
94514460600	0h 39m 13s	2274	57.99
92304793000	0h 00m 27s	12	26.67
91822312700	0h 09m 02s	503	22.47
95616399400	0h 19m 18s	412	21.35
91024811000	5h 02m 51s	8497	20.06
91049195200	0h 07m 26s	104	13.99
94979833800	1h 36m 12s	641	6.66

Tabelle 5.1: `Auswertung LMU München`

Die Angriffe finden zwar prinzipiell unabhängig von der Tageszeit statt, aber aus den Messdaten ist zu erkennen, dass sie Nachts und Mittags am effektivsten sind (siehe Abbil-

dung 5.3. Dies lässt den Schluss zu, dass während der Arbeitszeit durch niedrigere Netz-Kapazität Angriffe verlangsamt werden (um bis zu **89.57%**). Im Umkehrschluss könnte das aber auch bedeuten, dass diese stetigen Einbruchsversuche das Netz der LMU in nicht unerheblichem Maße zusätzlich belasten.

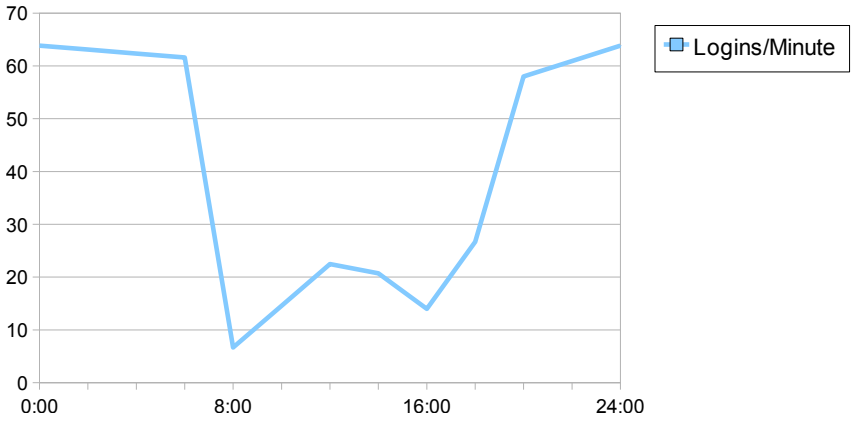


Abbildung 5.3: Logins/Minute als Hinweis auf Netzlast

5.2 atsec information security GmbH

Da dieses Projekt in Kooperation mit der *atsec information security GmbH* entstanden ist, bot sich ein Testeinsatz im dortigen Firmennetz regelrecht an. Mittlerweile wird das IDS, erweitert um viele neue Funktionen, in der Tat schon produktiv verwendet.

Das Unternehmen ist an mehreren Standorten in Europa, Asien und Amerika ansässig, deren interne Netze durch ein VPN miteinander verbunden sind. Aufgrund unterschiedlicher örtlicher Gegebenheiten ist eine differenzierte Aufstellung von Sensoren nötig. Abbildung 5.4 stellt den Aufbau exemplarisch dar.

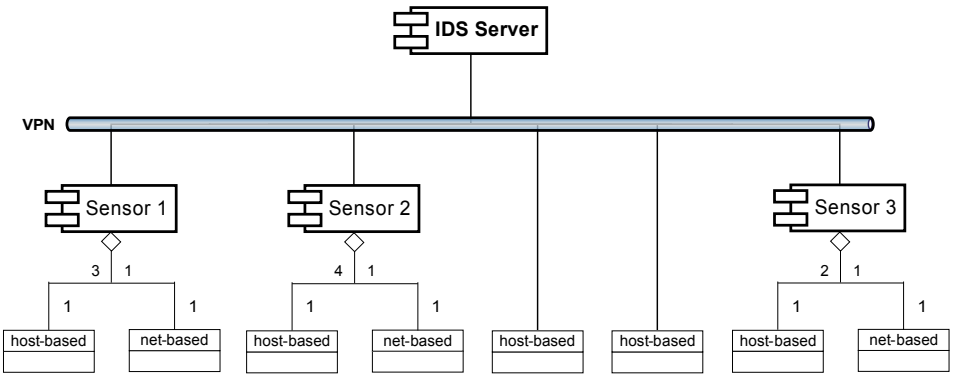


Abbildung 5.4: Exemplarische Aufstellung im Firmennetz

In diesem Szenario kann das IDS seine Vorzüge voll ausspielen. Ein simulierter verteilter Portscan auf unterschiedliche Teile des Netzes, der jeweils nur einige wenige Ports abtastet, wird ebenso erkannt wie komplexe Angriffsvektoren. Weiterhin wurden mehrere falsch konfigurierte Systeme entdeckt, die unnötigen internen Netzverkehr verursachten.

Ein wichtiger zu betrachtender Gesichtspunkt war ein möglichst geringes Datenaufkommen, um die Bandbreite des VPNs nicht übermäßig zu belasten. Dazu wurden Ansätze wie *Federated Databases* oder ein „verteilter IDS Server“ sowie Vorkorrelation auf den Sensoren geprüft.

Ebenfalls weiterentwickelt wurde das Reporting, das sich nun nicht mehr nur auf exemplarische Charts beschränkt, sondern einen kompletten Satz aussagekräftiger Metriken beinhaltet. Diese können – automatisch generiert oder ausgewählt vom Benutzer – als dynamisch erzeugter pdf-Report exportiert werden.

Desweiteren schreitet die Entwicklung einer *historischen Analyse* voran, wodurch Aufklärungs- oder Angriffsversuche über einen Zeitraum von mehreren Wochen oder Monaten hinweg erkannt werden sollen. Zur Verdeutlichung lassen sich einzelne überprüfte Ports pro Stunde oder Tag anführen, der in der Summe nach einer gewissen Zeit die wichtigsten Ports gescannt hätten, aber von herkömmlichen Einbruchserkennungs-Systemen bestenfalls als unabhängige, falsch geroutete Pakete erkannt werden würden.

Auch wenn diese fortgeschrittenen Funktionen nicht mehr Teil des Systementwicklungsprojekts sind, zeigt sich dennoch, dass es dem Anspruch einer generischen Implementierung mit Hinblick auf mögliche Erweiterungen vollkommen gerecht werden kann.

6 Zusammenfassung und Ausblick

Das Ziel dieses Systementwicklungsprojekts war der Entwurf und die Implementierung einer generischen Einbruchserkennung mit aussagekräftigem Reporting. Dazu wurde aus bekannten IDS-Ansätzen ein eigenes Vorgehen entwickelt, das deren Vorteile in sich vereinen konnte. Der zentrale Bestandteil beruhte dabei auf einer Analyse durch Korrelation. Um den Anforderungen wie breite Einsatzgebiete, Echtzeitfähigkeit oder Erkennung von Angriffsvektoren gerecht zu werden, musste die Implementierung strukturiert auf diese Ziele ausgelegt werden. Kapselung von Komponenten war dabei ebenso wichtig wie ein klares Konzept zur Behandlung und Verarbeitung von Ereignissen. Letztendlich mussten GUIs entwickelt werden, die sowohl die aktuelle Sicherheitslage wie auch längerfristige Entwicklungen mit Hilfe von Metriken darstellen konnten. Erste lauffähige Prototypen wurden sofort in verschiedenen Szenarien eingesetzt und kontinuierlich an die örtlichen Gegebenheiten angepasst.

Abschließend lässt sich sagen, dass das im Rahmen dieser Arbeit entwickelte System in vollem Umfang den Erwartungen entsprechen konnte. In den Testszenarien wurden falsch konfigurierte Systeme ebenso wie konkrete Angriffsversuche zuverlässig erkannt, außerdem lieferten die implementierten Metriken einen deutlichen Überblick über die veränderliche Sicherheitslage. Desweiteren ist das System generisch genug, um nahezu beliebige zusätzliche Informationsquellen anzubinden und das Einsatzgebiet dementsprechend zu erweitern.

Im Zuge einer Weiterentwicklung des Prototyps lässt sich eine fortgeschrittene Einbruchserkennung mittels historischer Analysen oder verallgemeinerter Erkennung von Angriffsvektoren anführen. Auch die Erweiterung des Reporting-Systems, beispielsweise um Patch- oder Qualitäts-Management sowie Risikoanalysen sind denkbar. Eine andere Richtung könnte mit reaktiven Komponenten eingeschlagen werden, die im Falle eines Angriffes nicht nur Alarme, sondern auch Gegenmaßnahmen auslösen können.

Zu guter letzt bleibt der Hinweis auf den mittlerweile produktiven Einsatz bei der *atsec information security GmbH*, bei dem die Möglichkeiten des hier vorgestellten Systems vollständig ausnutzt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[BSI, 2008] BSI (2008). IT-Sicherheitstrategie. http://www.bsi.bund.de/gshb/webkurs/gskurs/seiten/s2100.htm.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns*. Addison-Wesley Professional.

[ISO, 2005] ISO (2005). ISO/IEC 27001:2005. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42103.

[Kurose and Ross, 2007] Kurose, J. F. and Ross, K. W. (2007). *Computer Networking: A Top-Down Approach (4th Edition)*. Addison-Wesley, 4 edition.

[Lonvick, 2001] Lonvick, C. (2001). The BSD Syslog Protocol. http://www.ietf.org/rfc/rfc3164.txt.

[Meier, 2007] Meier, M. (2007). *Intrusion Detection effektiv!: Modellierung und Analyse von Angriffsmustern*. Springer Berlin Heidelberg.