# Effiziente Analyseverfahren für Intrusion-Detection-Systeme

Michael Meier, Sebastian Schmerl

Brandenburg Technische Universität Cottbus
Institut für Informatik, Postfach 10 13 44, 03013 Cottbus
{mm,sbs}@informatik.tu-cottbus.de

**Abstract:** In Ergänzung präventiver Mechanismen haben sich Intrusion-Detection-Systeme (IDS) als wichtiges Instrument für den Schutz informationstechnischer Systeme erwiesen. Die meisten heute eingesetzten IDS realisieren eine Signaturanalyse. Hierbei werden protokollierte Ereignisse mit vordefinierten Mustern (Signaturen), die auf IT-Sicherheitsverletzungen hindeuten, verglichen. Insbesondere für Systeme, die eine Erkennung komplexer, aus mehreren Schritten bestehender Attacken unterstützen, wurden bislang kaum Untersuchungen effizienter Analyseverfahren durchgeführt. Der Beitrag skizziert derzeit eingesetzte Analyseverfahren und stellt eine Reihe von Optimierungsstrategien vor, die unter Ausnutzung struktureller Eigenschaften von Signaturen Effizienzsteigerungen zu erreichen versuchen. Die Strategien wurden experimentell evaluiert und mit derzeitigen Signaturanalyseverfahren verglichen. Auf der Grundlage erhobener Messwerte konnten signifikante Leistungsverbesserungen nachgewiesen werden.

## 1 Einführung

Intrusion-Detection-Systeme (IDS) haben sich als ein wichtiges Instrument für den Schutz informationstechnischer Ressourcen erwiesen. In Ergänzung präventiver Sicherheitsmechanismen führen sie eine automatische Erkennung und ggf. auch eine Abwehr von IT-Sicherheitsverletzungen durch. Für die Erkennung von Sicherheitsverletzungen existieren zwei grundlegende Strategien, die *Signaturanalyse* (*Misuse Detection*) und die *Anomalieerkennung* (*Anomaly Detection*). Erstere untersucht Protokoll- bzw. Audit-Daten nach Mustern bekannter Sicherheitsverletzungen, den Signaturen. Anomalieerkennung untersucht Nutzer- oder Systemaktivitäten auf Abweichungen von vordefinierten oder ermittelten Profilen und kennzeichnet diese Abweichungen als Sicherheitsverletzungen. Der Vorteil signaturbasierter Verfahren liegt in ihrer höheren Erkennungsgenauigkeit. Ferner sind derartige Systeme einfacher zu konfigurieren. Anomalieerkennung bietet den Vorteil unbekannte Sicherheitsverletzungen erkennen zu können, führt aber aufgrund geringer Erkennungsgenauigkeit häufig zu unakzeptabeln Fehlalarmraten. Darüber hinaus signalisieren diese Systeme lediglich Anomalien, von denen häufig nicht ohne weiteres auf eine konkrete Sicherheitsverletzung geschlossen werden kann.

In Abhängigkeit vom Ablauf und damit vom erforderlichen Aufwand zu ihrer Erkennung können Attacken in zwei Klassen unterteilt werden. *Single-Step-Attacken* bezeichnen Sicherheitsverletzungen, die auf Grundlage eines einzelnen Audit-Eintrags erkennbar sind. Signaturen zur Erkennung derartiger Attacken beschreiben charakteristische Byte-Sequenzen, deren ggf. kombiniertes Auftreten in einem Audit-Record auf eine Si-

cherheitsverletzung hindeutet. *Multi-Step-Attacken* hingegen bezeichnen Sicherheitsverletzungen zu deren Erkennung mehrere Protokolleinträge in Zusammenhang gebracht und auf charakteristische Merkmale untersucht werden müssen. *Single-Step-IDS* bezeichnen Systeme, die in der Lage sind Single-Step-Attacken zu erkennen. Die Analyseverfahren dieser Systeme basieren im Wesentlichen auf String-Vergleiche. Die Beschränkungen dieser Systeme überwinden *Multi-Step-IDS*, die zusätzlich Multi-Step-Attacken erkennen und dazu deutlich komplexere Analyseverfahren verwenden.

Eine Herausforderung, der sich alle IDS gegenübersehen, ist die wachsende Leistungsfähigkeit von Netzwerken und Endsystemen, mit der Steigerungen des Aufkommens an Protokolldaten einhergehen. Ferner führt die wachsende Komplexität der IT-Systeme zu neuen Verwundbarkeiten und damit Angriffswegen, so dass die zu analysierenden Signaturen zunehmen. Damit gewinnt die Effizienz der von IDS eingesetzten Analyseverfahren an Bedeutung. Bereits heute werden in Lastsituationen Protokolldaten vom IDS verworfen oder die Erkennung von Sicherheitsverletzungen wird signifikant verzögert, so dass Gegenmaßnahmen nur noch eingeschränkt möglich sind. Im Hinblick auf diese Anforderungen werden verschiedene Ansätze verfolgt. Bspw. wird die Einbruchserkennung auf der Grundlage kompakterer, weniger detaillierter Protokolldaten untersucht [1, 2, 3]. Darüber hinaus wurden verschiedene optimierte Analyseverfahren für signaturbasierte Single-Step-IDS entwickelt [4, 5]. Bisher wenig Aufmerksamkeit wurde hingegen der Optimierung der Analyseverfahren von Multi-Step-IDS gewidmet. Dieser Beitrag skizziert derzeit eingesetzte Standard-Analyseverfahren von Multi-Step-IDS. Darüber hinaus werden verschiedene Optimierungsstrategien, denen die Ausnutzung von strukturellen Eigenschaften von Signaturen zugrunde liegt, vorgestellt und bewertet.

Abschnitt 2 skizziert zunächst einen Ansatz zur Modellierung und Beschreibung komplexer Multi-Step-Signaturen, bevor Abschnitt 3 existierende Analyseansätze von Multi-Step-IDS diskutiert. In Abschnitt 4 werden verschiedene Optimierungsstrategien für derartige Analyseverfahren vorgestellt. Abschnitt 5 stellt Ergebnisse von durchgeführten Laufzeitmessungen vor und vergleicht diese mit existierenden Ansätzen. Der Beitrag schließt mit einer Zusammenfassung.

## 2 Modellierung komplexer Angriffssignaturen

Bei der Durchführung einer Sicherheitsverletzung generierte Audit-Datensätze stellen die *Manifestierung* (Spuren) der Attacke dar. Die *Signatur* einer Attacke beschreibt die Kriterien (Muster), anhand derer die Spuren einer Attacke in einem Audit-Datenstrom identifiziert werden können. Es ist möglich, dass mehrere Attacken eines Typs gleichzeitig und unabhängig voneinander fortschreiten, beispielsweise indem sie zeitgleich von verschiedenen Angreifern durchgeführt werden. Deshalb ist es erforderlich, zwischen verschiedenen Instanzen einer Attacke, deren Manifestierungen sich in der Ausprägung einzelner Merkmale z.B. dem Benutzernamen unterscheiden, zu differenzieren. Eine *Signaturinstanz* beschreibt die Menge der Kriterien, die eindeutig die Manifestierung einer Attackeninstanz im Audit-Datenstrom identifizieren.

Im Kontext komplexer, aus mehreren Schritten bestehender Attacken, ist es erforderlich eine Reihe verschiedener Aspekte in den Signaturen zu beschreiben. Eine ausführliche

Diskussion der verschiedenen Aspekte ist in [6] zu finden. Als hilfreiches Werkzeug zur Modellierung komplexer Signaturen hat sich ein auf gefärbten Petrinetzen basierender Modellierungsansatz [7] erwiesen. Ausgehend von den positiven Erfahrungen mit diesem Modellierungsansatz wurde die Signaturbeschreibungssprache EDL (Event Description Language) entwickelt, die sich an dem Petrinetz-basierten Modellierungsansatz orientiert und gleichzeitig alle in [6] identifizierten Aspekte von Signaturen auszudrücken vermag. Im Folgenden werden die grundlegenden und zum Verständnis der weiteren Ausführungen notwendigen Aspekte der Sprache EDL eingeführt. Eine ausführliche Beschreibung findet sich in [8].

### 2.1 Überblick der Beschreibungskonstrukte

Signaturbeschreibungen in EDL bestehen aus *Plätzen* und *Transitionen*, die mit gerichteten *Kanten* verbunden sind. Plätze beschreiben Teilzustände des Systems, die eine entsprechende Attacke durchläuft. Transitionen repräsentieren Zustandsübergänge und stellen die einzelnen Ereignisse, z.B. sicherheitskritische Aktionen, dar, aus denen eine Attacke besteht. Jeder Platz ist mit mindestens einer Transition und jede Transition mit mindestens einem Platz über eine gerichtete Kante verbunden. Plätze, von denen eine Kante zu einer Transition *t* führt, stellen die *Vorplätze* dieser Transition *t* dar. Die Plätze, zu denen eine gerichtete Kante von der Transition *t* führt, sind *Nachplätze* der Transition *t*. *Token* sind die dynamischen Elemente von EDL-Signaturen. Während der laufenden Analyse einer Signatur repräsentieren Token konkrete Signaturinstanzen. Dazu werden Token, analog zu gefärbten Petrinetzen, mit Werten belegt.

### 2.2 Plätze

Um die relevanten Teilzustände des Systems bei einer Attacke zu beschreiben, sind Plätze durch eine Menge von *Merkmalen* und einen *Platztyp* gekennzeichnet. Die Merkmale beschreiben die Eigenschaften von Token, die sich auf diesem Platz befinden. EDL unterscheidet zwischen den vier Platztypen *Initial-*, *Interior-*, *Escape-* und *Exit-Platz*. *Inital-Plätze* sind die Startplätze einer Signatur und sind zum Analysebeginn mit einem initialen Token belegt. Initiale Plätze besitzen keine Merkmale. Jede EDL-Signatur besitzt genau einen *Exit-Platz*, der den Endplatz der Signatur darstellt. Wenn ein Token diesen Platz erreicht, wurde durch die Signatur eine Manifestierung der entsprechenden Attacke im Audit-Datenstrom erkannt. *Escape-Plätze* beschreiben den Abbruch der Verfolgung einer Attackeninstanz. Sie werden erreicht, wenn Ereignisse auftreten, die eine Vervollständigung der Attackeninstanz unmöglich machen. Escape-Plätze besitzen keine Merkmale. Token, die einen Escape-Platz erreichen, werden gelöscht. Alle anderen Plätze sind vom Typ *Interior* und können wie Exit-Plätze beliebige Merkmale besitzen.

Zur Illustration zeigt Abb. 1 eine einfache Signatur mit den Plätzen $P_1$ bis $P_4$. $P_1$ besitzt als initialer Platz keine Merkmalsdefinitionen und somit enthält das Token auf $P_1$ keine Wertebelegungen. Der Platz $P_2$ besitzt das Merkmal *UserID* vom Datentyp Int. Die zwei Token auf $P_2$ besitzen unterschiedliche Wertebelegungen (*1066* bzw. *1080*) für das Merkmal von *UserID*. Platz $P_3$ definiert zwei Merkmale. Die zugehörigen Token auf $P_3$ enthalten die dargestellten Wertebelegungen. Der Exit-Platz $P_4$ besitzt die Merkmale *OpenFile* und *TimeStamp*. Das Token auf $P_4$ belegt diese mit *".mail"* und *1091*.

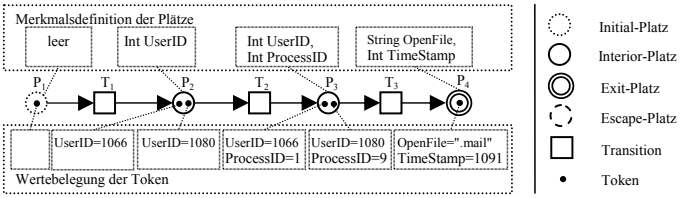


Abb. 1: Merkmale von Plätzen

## 2.3 Transitionen

Transitionen repräsentieren die Ereignisse, die Zustandsübergänge der Signaturinstanzen bewirken. Eine Transition wird charakterisiert durch Vorplätze, Nachplätze, Ereignistyp, Bedingungen, Merkmalsbindungen, Konsumtivität und Aktionen. Die *Vorplätze* einer Transition beschreiben den erforderlichen Zustand des Systems vor der Auslösung der Transition. Die *Nachplätze* charakterisieren den durch die Transition angepassten Systemzustand. Ein Übergang zwischen den Systemzuständen setzt ein sicherheitsrelevantes Ereignis voraus. Daher ist jede Transition mit einem *Ereignistyp* assoziiert. Des Weiteren kann für einen Zustandsübergang die Erfüllung zusätzlicher *Bedingungen* erforderlich sein, mit denen einerseits weitere Eigenschaften des Ereignisses oder bestimmte Zusammenhänge zwischen dem aktuellen Zustand und dem Ereignis gefordert werden. Das heißt es können Bedingungen formuliert werden, die fordern, dass bestimmte Merkmale des Ereignisses (z.B. Benutzername) mit bestimmten Werten (z.B. root) belegt sein müssen. Außerdem können über Bedingungen Beziehungen zwischen Ereignismerkmalen und Merkmalen von Token auf Vorplätzen (z.B. gleicher Wert) beschrieben werden.

Durch Schalten einer Transition entstehen auf den Nachplätzen neue Token, die den neuen Systemzustand kennzeichnen. Um die Werte der Merkmale der neuen Token auf den Nachplätzen zu definieren, besitzt eine Transition *Merkmalsbindungen*. Diese können einfache Zuweisungen oder komplexe Ausdrücke, parametrisiert von Konstanten, Referenzen auf Ereignismerkmale bzw. Vorplatzmerkmale, sein. Die *Konsum-Eigenschaft* (vgl. [6]) einer Transition steuert das Verbleiben der transitionsaktivierenden Token auf den Vorplätzen nach dem Schalten der Transition. Die Eigenschaft wird für verschiedene Vorplätze separat definiert. Nur im *consuming*-Fall wird das Token beim Schalten der Transition vom Vorplatz entfernt. Transitionen können *Aktionen* zugeordnet werden, die beim Schalten der Transition ausgeführt werden. Typische Aktionen sind die Generierung von Alarmmeldungen oder die Initiierung von Gegenmaßnahmen.

Abb. 2 illustriert die Eigenschaften von Transitionen. $T_1$ besitzt zwei Bedingungen. Die erste fordert, dass Ereignis *E* das Merkmal *Typ* mit dem Wert *FileCreate* belegt. Die zweite Bedingung bezieht sich mit *P1.UserID* auf das Merkmal *UserID* des Platzes $P_1$ und *EUserID* referenziert das gleichnamige Merkmal des Ereignistyps *E*. Diese Bedingung fordert, dass der Wert des Tokenmerkmals *UserID* auf $P_1$ mit dem Wert des Ereignismerkmals *EUserID* übereinstimmt. $T_1$ besitzt zwei Merkmalsbindungen: Dem Merkmal *UserID* des neuen Tokens auf $P_2$ wird der Wert des Merkmals *UserID* des auslösenden Tokens von $P_1$ zugewiesen. Das Merkmal *Name* des neuen Tokens auf $P_2$ wird mit dem Wert des Ereignismerkmals *EName* belegt. $T_1$ besitzt eine Aktion, einen parametrisierten Aufruf der Funktion *"Warn"*.

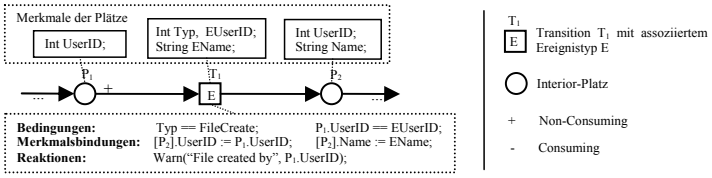


Abb. 2: Eigenschaften von Transitionen

# 3 Existierende Analyseverfahren

Existierende Analyseverfahren für Multi-Step-IDS lassen sich grob in zwei Kategorien unterteilen. Verfahren der ersten Kategorie übersetzen jede Signaturbeschreibung in ein separates Programmmodul. Die andere Kategorie von Systemen setzt Standardexpertensysteme zur Analyse ein.

## 3.1 Programmmodule zur Analyse von Signaturen

Vertreter dieser Kategorie sind die STAT Toolsuite [9] und das IDIOT-IDS [10]. Während STAT die Zustands-Transitions-basierte Signaturbeschreibungssprache STATL [11] zur Beschreibung von Signaturen verwendet, werden Signaturen für das IDIOT-IDS als Coloured Petri Net Automatons [10] spezifiziert. Beide Systeme bieten einen entsprechenden Compiler, der die Signaturspezifikationen in C++-Klassenmodule übersetzt. Dabei wird für jede Signatur eine separate Klasse angelegt und in jeweils eine Shared-Library übersetzt. Die Systeme verwalten intern eine Liste verfügbarer Signaturbibliotheken. Zur Laufzeit werden die Protokolldatensätze nacheinander an die einzelnen Signaturbibliotheken übergeben, welche den Datensatz entsprechend der spezifizierten Kriterien analysieren. Bereits bei Betrachtung dieses grundlegenden Analyseablaufs kann festgestellt werden, dass verschiedene Signaturen und Signaturinstanzen jeweils unabhängig voneinander analysiert werden. Somit werden vorhandene Redundanzen nicht vermieden, was die Effizienz des Ansatzes infrage stellt.

## 3.2 Expertensystem-basierte Signaturanalyse

Charakteristisch für Expertensysteme ist die Trennung zwischen Anwendungswissen und allgemeinen Problemlösungsstrategien. Anwendungswissen wird in Form von Regeln beschrieben, die auf Fakten im Working-Memory des Expertensystems angewendet werden. Die Problemlösung obliegt vorgegebenen Inferenzmechanismen. Aufgrund dieser Trennung sind Expertensysteme flexibel einsetzbar und leicht erweiterbar (vgl. [12]).

Vertreter dieser Kategorie sind die IDS EMERALD [13], CMDS [14] und AID [15]. Während EMERALD auf der Expertensystemshell P-Best [16] basiert, verwendet CMDS das Expertensystem CLIPS [17]. AID wurde unter Verwendung der kommerziellen Expertensystemshell RT-Works [18] implementiert. Vorteil der Verwendung von Expertensystemen ist die einfache Implementierung und Verwendung optimierter Inferenzalgorithmen.

Signaturen für diese Systeme werden direkt als Regeln beschrieben oder aus entsprechenden Signaturbeschreibungssprachen in Regeln übersetzt. Im Folgenden wird eine mögliche Abbildung von EDL-Signaturen auf Regeln skizziert. Eine ausführliche Beschreibung findet sich in [19]. Fakten des Working-Memory repräsentieren das aktuell zu analysierende Audit-Record sowie die existierenden Token. Transitionen werden auf Regeln abgebildet, die im IF-Teil das Vorhandensein entsprechender Token und eines passenden Audit-Records sowie angegebene Bedingungen prüfen. Der Then-Teil verändert, erstellt oder löscht Token und führt Aktionen aus. Plätze werden als Merkmale der Token dargestellt. Somit wird die Bewegung eines Tokens durch Setzen des Platzmerkmals des entsprechenden Tokenfakts auf den neuen Platznamen umgesetzt. Der Analysezyklus des Systems sieht vereinfacht wie folgt aus: Einfügen des aktuellen Audit-Records als Fakt in den Working-Memory. Überprüfen aller Regeln und ggf. ausführen. Ersetze das aktuelle Audit-Record durch das nächste und prüfe erneut alle Regeln usw.

Expertensysteme setzen optimierte Match-Algorithmen ein. Der bekannteste und meist verwendete Algorithmus ist der RETE-Algorithmus [20]. Ihm liegen zwei Hauptideen zu Grunde: Vermeidung des Iterierens über die Regelmenge und Vermeidung des Iterierens über den Working-Memory. Der ersten Idee liegt die Beobachtung zugrunde, dass der größte Teil der Laufzeit zur Überprüfung der Regelbedingungen benötigt wird. Optimiert wird hier in dem die Regelbedingungen in einen Datenflussgraphen, das sog. RETE-Netzwerk, abgebildet werden. Dadurch wird erreicht, dass Bedingungen, die in mehreren Regeln auftreten, nicht mehrfach evaluiert werden. Diese Technik ist mit Common Subexpression Elemination [21] vergleichbar. Der zweiten Idee liegt die Annahme zugrunde, dass sich die Fakten nur relativ selten ändern. Dies wird ausgenutzt, indem Fakten im RETE-Netzwerk an den Bedingungsknoten gespeichert werden, deren Bedingung sie erfüllen. Fortlaufend müssen dann nur die Fakten überprüft werden, die sich geändert haben. Die Kosten für diesen Ansatz spiegeln sich darin wieder, dass Fakten die gelöscht werden, aus dem RETE-Netzwerk entfernt werden müssen. Dazu müssen die Bedingungen bzgl. dieses Faktes erneut evaluiert werden, um die Bedingungsknoten zu identifizieren in denen der Fakt gespeichert ist.

Während sich die zweite Annahme in vielen klassischen Anwendungsfeldern von Expertensystemen als zutreffend erweist und zu deutlichen Performanzsteigerungen führt, erscheint die Gültigkeit der Annahme im Kontext von signaturbasierter Intrusion Detection zweifelhaft: Der Fakt Audit-Record ändert sich in jedem Zyklus, d.h. der alte Fakt wird gelöscht und der neue eingefügt. Spezifisch für den Anwendungsfall der Signaturanalyse ist weiterhin, dass jede Regel mindestens eine Bedingung enthält, die Bezug auf den Fakt Audit-Record nimmt. Das wirft die Frage auf, ob der Nutzen des RETE-Verfahrens seine Kosten übersteigt und stellt die Motivation für die Entwicklung dedizierter Optimierungsstrategien für Signaturanalysesysteme dar.

## 4 Optimierungansätze

Ein naiver Analyseprozess prüft für jedes eingehende Ereignis *X* alle Transitionen aller Signaturen. Beim Transitionstest wird geprüft, ob der Typ des Ereignisses *X* dem assoziierten Typ der Transition entspricht. Anschließend werden für jede Tokenkombination auf den Vorplätzen die Bedingungen der Transition in Zusammenhang mit dem Ereignis

*X* evaluiert. Im laufenden Betrieb der Analyseeinheit nehmen die zu testenden Tokenkombinationen zu, wodurch der Analyseaufwand für ein einzelnes Ereignis ansteigt. In diesem Abschnitt wird der skizzierte Analyseprozess schrittweise erweitert und es werden fünf verschiedene Optimierungsstrategien vorgestellt.

### 4.1 Strategie 1: Typ-Indizierung

Beim Auftreten eines Ereignisses muss für alle in Signaturen spezifizierten Transitionen überprüft werden, ob das aufgetretene Ereignis dem Ereignistyp der Transition entspricht. Diese Überprüfungen können vermieden werden, in dem eine Tabelle verwendet wird, in der jedem Ereignistyp die Menge der Transitionen zugeordnet wird, die mit diesem Ereignistyp assoziiert sind. Somit kann in konstanter Zeit die Menge der Transitionen ermittelt werden, für die das aktuelle Ereignis relevant ist.

### 4.2 Strategie 2: Instanzunabhängige Bedingungsüberprüfung

Eine Transition schaltet, wenn alle Bedingungen der Transition erfüllt sind. Diese Bedingungen lassen sich in Inter- und Intra-Ereignisbedingungen unterscheiden. Während erstere Merkmale des Ereignisses mit Merkmalen von Token in Beziehung setzen (zweite Bedingung in Abb. 2), beziehen sich Intra-Ereignisbedingungen lediglich auf die Merkmalsbelegungen des auslösenden Ereignisses (erste Bedingung in Abb. 2). Dementsprechend können Intra-Ereignisbedingungen unabhängig von den Token auf den Vorplätzen der Transition überprüft werden. Deshalb genügt es, Intra-Ereignisbedingungen für das aktuelle Ereignis einmal zu prüfen, und nur wenn diese erfüllt sind, die Inter-Ereignisbedingungen für alle Kombinationen von Token auf den Vorplätzen zu testen.

### 4.3 Strategie 3: Wertebasierte Tokenindizierung

Bei der Analyse einer durch Strategie 1 und 2 selektierten Transition muss über alle Token-Kombinationen auf den Vorplätzen iteriert werden, um die Kombinationen zu identifizieren, die alle Inter-Ereignisbedingungen erfüllen. Durch die Verwendung von Wertetabellen können an Transitionen spezifizierte Vergleichsoperationen zur Reduktion der zu testenden Token-Kombinationen benutzt werden. Eine Wertetabelle bildet die Werte eines Platz-Merkmals in eine Menge von Token ab. Dadurch kann in konstanter Zeit bestimmt werden, welche Token auf dem Platz das Merkmal mit einem bestimmten Wert belegen. Fordert bspw. eine Bedingung die Gleichheit der Werte vom Ereignismerkmal *A* und dem Tokenmerkmal *B* auf Vorplatz *P*, so kann durch einen einzelnen Zugriff mit dem Wert von *A* in die Wertetabelle von *B* die Menge *X* der Token auf *P* selektiert werden, die diese Bedingung erfüllen. Falls eine weitere Gleichheitsbedingung zwischen dem Ereignis- und einem Tokenmerkmal auf Platz *P* existiert, wird wiederum durch einen Zugriff eine zweite Menge *Y* selektiert. Der Schnitt der Mengen *X* und *Y* enthält somit die potentiell auslösbaren Token von Platz *P*. Dieses Vorgehen liefert anhand von Gleichheitsbedingungen effizient die Menge passender Token. Der Vorgang ist nicht auf Gleichheitsbedingungen beschränkt, sondern kann bei sämtlichen Vergleichsoperationen verwendet werden. Dies erfordert eine sortierte Ablage der Schlüsselwerte in den Wertetabellen und effiziente Mechanismen zur Wertebereichsselektion.

Der Ansatz ist auch bei Vergleichsbedingungen zwischen Merkmalen von Vorplätzen realisierbar. Abb. 3 zeigt dies exemplarisch. Hier wird gefordert, dass Merkmal *UserID* auf Platz *P* und Merkmal *Owner* auf *Q* die gleichen Werte besitzen. Ferner soll *P.Host ==Q.Host* gelten. Die Wertebelegung der Token und die Wertetabellen sind in Abb. 3 dargestellt. Für die erste Bedingung wird mit den Werten aus der Tabelle *P.UserID* auf die Tabelle *Q.Owner* zugegriffen. Für jede Übereinstimmung werden platzweise Tokenmengen selektiert. Aus der platzweisen Vereinigung dieser Mengen resultiert die Menge der Token, die die erste Bedingung erfüllen. Für die zweite Bedingung wird analog vorgegangen. Der platzweise Schnitt, der aus den Bedingungen resultierenden Tokenmengen, erfüllt beide Bedingungen (im Bsp. Token 1 auf *P* und Token 3 auf *Q*).

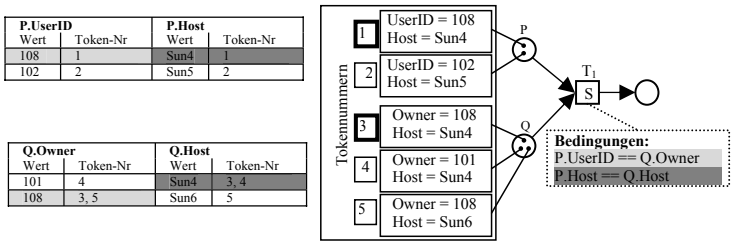


Abb. 3: Wertebasierte Tokenindizierung

### 4.4 Strategie 4: Identifikation gemeinsamer Teilausdrücke

Bei der Analyse eines Ereignisses werden die Transitionsbedingungen überprüft. Unterschiedliche Transitionen können in ihrem Bedingungsblock gleiche Ausdrücke bzw. Teilausdrücke enthalten (Common Subexpressions). Um eine mehrfache Analyse dieser Ausdrücke zu vermeiden, werden die Intra-Ereignisbedingungen aller Transitionen auf gemeinsame Ausdrücke untersucht. Bei Inter-Ereignisbedingungen werden gemeinsame Ausdrücke nur innerhalb einer Transition identifiziert. Evaluierte Werte der gemeinsamen Ausdrücke werden gespeichert und bei einer erneuten Analyse des gleichen Ausdrucks verwendet. Die gespeicherten Werte behalten bei Intra-Ereignisbedingungen ihre Gültigkeit für den Zeitraum der Abarbeitung eines Ereignisses. Dagegen ist der Gültigkeitszeitraum bei Inter-Ereignisbedingungen auf eine Tokenkombination beschränkt, da andere Tokenkombinationen andere Wertebelegungen repräsentieren. Standardtechniken zur Identifizierung von gemeinsamen Ausdrücken werden in [21] diskutiert.

### 4.5 Strategie 5: Kostenbasierte Bedingungspriorisierung

Aus der Vielzahl der zu analysierenden Ereignisse löst nur ein geringer Teil eine Transition aus. Die Bedingungen einer Transition werden somit hauptsächlich negativ ausgewertet. Die Reihenfolge der Bedingungsüberprüfung sollte somit auf Misserfolg optimiert erfolgen. Davon ausgehend sollten Teilbedingungen mit geringeren Laufzeiten vor aufwendigern analysiert werden, damit bei der Nichterfüllung die aufwendigeren Teilbedingungen nicht evaluiert werden müssen. Die Bedingungspriorisierung kann statisch oder dynamisch erfolgen. Die statische Priorisierung kategorisiert die Bedingungen auf Basis von Laufzeitabschätzungen der verwendeten Operationen. Bei der dynamischen

Bedingungspriorisierung wird davon ausgegangen, dass die zu analysierenden Ereignisse stark abhängig von den Nutzeraktivitäten auf dem überwachten IT-System sind. Dabei schränken die Nutzeraktivitäten den Typ und die Wertebereiche der Merkmalsbelegungen der Ereignisse indirekt ein. Somit sind in Abhängigkeit von den aktuell vorherrschenden Aktivitäten bestimmte Bedingungen wahrscheinlicher erfüllt als andere. Weiterhin beeinflussen die Wertebelegungen der auftretenden Ereignisse die Evaluierungszeit von komplexeren Ausdrücken. Um die Analyse diesen Umständen anzupassen, werden die realen Laufzeiten und die Negativquote der Bedingungen in bestimmten Zeitintervallen erhoben. Der Quotient von Laufzeit und Negativquote einer Bedingung liefert ein Maß dafür wie effizient die Bedingung eine Ereignis- bzw. Token-Kombination verwirft und wird zur Priorisierung der Bedingungen verwendet. Regelmäßige Aktualisierungen der Messwerte passen die Analyse an das aktuelle Systemverhalten an.

# 5 Laufzeitmessungen

Um einen Eindruck vom Ausmaß der erreichbaren Leistungsteigerungen durch die vorgestellten Optimierungsstrategien zu erhalten, wurden diese im Analysemodul SAM (Signature Analyse Module) prototypisch implementiert und vermessen. In diesem Abschnitt wird die Laufzeit von SAM mit den Laufzeiten der Analysewerkzeuge STAT und CLIPS (als Vertreter für Expertensysteme) verglichen.

## 5.1 Testumgebung

Als Testszenarien wurden eine Shell-Link-Attacke, eine SuidScript-Attacke und eine Login-Attacke verwendet. Die zugehörigen Signaturen liegen semantisch äquivalent für STAT, CLIPS und SAM vor. Im Weiteren werden die Attacken kurz erläutert und die EDL-Signaturen graphisch veranschaulicht.

Die **Shell-Link-Attacke** nutzt den speziellen Shell-Mechanismus und den SUID-Mechanismus aus. Wird auf ein Shell-Skript ein Link angelegt, dessen Name mit "-" beginnt, so ist es möglich, über einen Aufruf des Links eine interaktive Shell zu erzeugen. Bei älteren Shell-Versionen bestand die Möglichkeit, einen Link auf SUID-Shell-Skripte zu erzeugen und durch dessen Aufruf eine interaktive Shell mit Privilegien des Shell-Skript-Eigentümers zu erzeugen. Abb. 4 veranschaulicht die zugehörige EDL-Signatur.

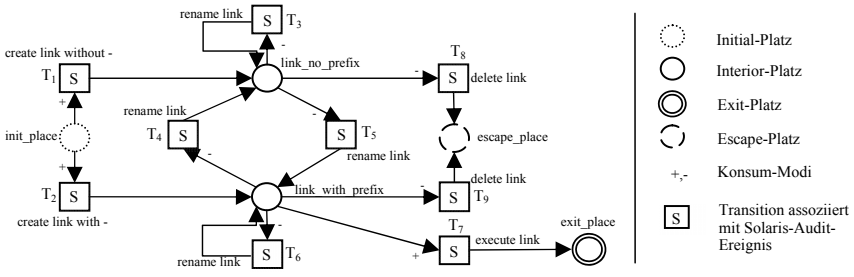


Abb. 4: Schema der EDL-Signatur für die Shell-Link-Attacke

Die **SuidScript-Attacke** nutzt die Umgebungsvariable PATH und den SUID-Mechanismus aus. Die PATH Variable legt für einen Nutzer den Suchpfad für ausführbare Dateien fest. Steht am Anfang ein Verzeichnis, das vom Angreifer beschrieben werden kann und existiert ein SUID-Root-Shell-Skript, das ein Kommando ohne Angabe des vollständigen Pfades aufruft, so kann vom Angreifer ein gleichnamiges Programm angelegt werden, welches dann bei der Ausführung des Skriptes mit den Rechten von root gestartet wird. Abb. 5 zeigt die schematische EDL-Signatur.

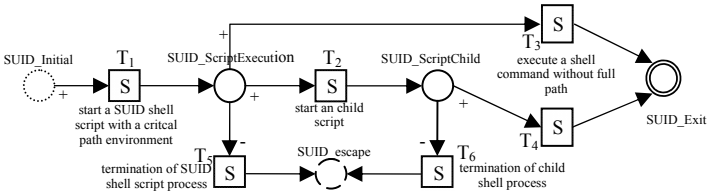


Abb. 5: Schema der EDL-Signatur für die SUID-Script-Attacke

Die **Login-Attacke** stellt einen Brute-Force-Angriff dar. Dabei wird versucht durch mehr oder weniger sinnvolles Probieren das Passwort eines Benutzers zu raten. Bei Erkennung drei fehlerhafter Anmeldeversuche innerhalb von 60 Sekunden kann von einem Angriff ausgegangen werden. Abb. 5 stellt die zugehörige EDL-Signatur dar.

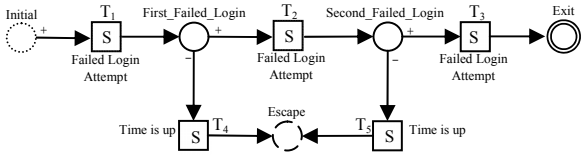


Abb. 6: Schema der EDL-Signatur für die Login-Attacke

Diese drei Angriffe wurden in einer Audit-Trail aufgezeichnet. Um die Komplexität des Tests zu erhöhen, wurde diese Audit-Trail 10.000 mal vervielfältigt. Durch Umbenennung der involvierten Datei-, Verzeichnisnamen und Nutzerkennungen wurden die Angriffe entkoppelt. Als Resultat steht eine Audit-Trail mit 10.000 unabhängigen Angriffen bzw. 110.000 Ereignissen zur Verfügung. Dabei resultiert jede Attacke in mindestens einer Signaturinstanz, die während der gesamten weiteren Analyse berücksichtigt werden muss. Das Testszenario stellt einen Worst-Case-Fall dar, der in der Realität selten auftreten wird, jedoch eine Beurteilung des Leistungsvermögens in Lastsituationen erlaubt.

### 5.2 Messergebnisse

Die Testumgebung für STAT bestand aus einem Host mit 2 UltraSPARC-III+ a 900 MHz CPUs und 4GB RAM. CLIPS und SAM wurden unter Windows 2000 auf einem Pentium-III mit 300MB RAM vermessen. Um die Messungen vergleichen zu können, werden statt absoluter Laufzeiten die Änderungen der Laufzeiten in Abhängigkeit von der Zahl bereits analysierter Ereignisse betrachtet. Gemessen wurde jeweils die Laufzeit zur Analyse von 1.000 Ereignissen. Das heißt es wurde ermittelt wie viel Zeit die Analyse der Ereignisse 1 – 1.000, 1.001 – 2.000, 2.001 - 3.000 usw. benötigt. In Abb. 7 werden die Steigerungen der ermittelten Zeiten von STAT, CLIPS und SAM dargestellt.

Bspw. benötigt STAT für die Ereignisse 20.001 - 21.000 ca. das Fünffache der Zeit wie für Analyse der Ereignisse 1 – 1.000. Hinter den Testkandidaten ist die zur Verarbeitung von 40.000 Ereignissen benötigte absolute Laufzeit dargestellt.

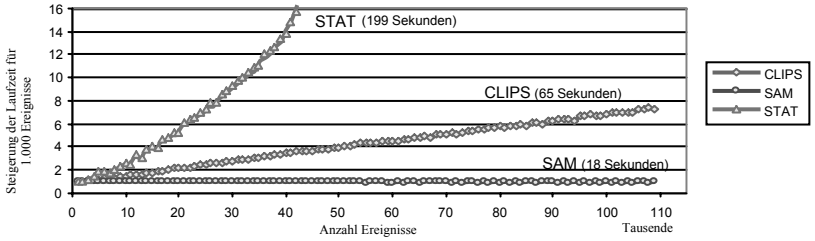


Abb. 7: Steigerung der Laufzeiten im Vergleich

Der Analyseaufwand von STAT steigt mit der Anzahl analysierter Ereignisse polynomial an. Dieser Umstand ist auf die instanzunabhängige Auswertungsmethodik zurückzuführen, mit steigender Anzahl der Ereignisse erhöht sich die Anzahl der zu testenden Instanzen. In STAT werden weder strukturelle Zusammenhänge in den Signaturen noch in den Signaturinstanzen berücksichtigt. CLIPS zeigt dagegen einen geringeren Anstieg. Dies ist auf den Rete-Algorithmus zurückzuführen, zwar werden auch hier die Instanzen unabhängig voneinander betrachtet, aber es werden identische Bedingungen nicht mehrmals evaluiert. Die Messergebnisse von SAM zeigen im Gegensatz zu STAT und CLIPS einen konstanten Analyseaufwand mit steigender Eingabemenge. Dieser Umstand ist hauptsächlich auf wertebasierte Tokenindizierung zurückzuführen. Durch die Ausnutzung der geforderten Gleichheitsbedingungen an den Transitionen können effizient die zu analysierenden Token-Kombinationen auf den Vorplätzen der Transitionen auf die potentiell nützliche Kombination reduziert werden. Die Leistungsgewinne der einzelnen Strategien sowie deren Kombinationen werden in [8] ausführlich diskutiert.

Um zu untersuchen, inwieweit die Optimierungen von SAM durch einen übermäßigen Speicherverbrauch erkauft werden, wurde während der Tests der benötigte Hauptspeicher ermittelt. Zwar wächst der Speicherverbrauch von SAM aufgrund der Wertetabellen mit den analysierten Ereignissen, er blieb jedoch während des gesamten Tests deutlich unter den Werten von CLIPS und STAT. Tab. 1 stellt die erhobenen Spitzenwerte dar.

| **Testkandidat** | **Hauptspeicherverbrauch** |
|---|---|
| SAM | 36 MB |
| CLIPS | 58 MB |
| STAT | 638 MB |

Tab. 1: Hauptspeicherbelastung durch die Testkandidaten

## 6 Zusammenfassung

Dieser Beitrag skizzierte existierende Analyseverfahren für Multi-Step-IDS und diskutierte die Leistungsfähigkeit dieser Verfahren. Ausgehend von der Anwendung einer Petrinetz-basierten Modellierung komplexer Attacken, wurden verschiedene Eigenschaften von Signaturen identifiziert, die zur Optimierung des Analyseablaufs herangezogen

werden können. Fünf Optimierungsstrategien wurden vorgestellt und unter Verwendung einer erstellten Implementierung experimentell evaluiert. Im Vergleich mit den existierenden Analyseverfahren führten die entwickelten Optimierungsstrategien zu signifikanten Leistungsverbesserungen. Dies zeigt, dass durch die Entwicklung spezialisierter Signaturanalysealgorithmen deutliche Laufzeitverbesserungen gegenüber Standardverfahren wie Expertensystemen erreicht werden können.

## 7 Literaturverzeichnis


[1] Cisco Systems Inc.: NetFlow Services and Applications. White Paper. 15 Jul. 2002, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm

[2] J. McHugh: Set, Bags and Rock and Roll – Analyzing Large Datasets of Network Data. In: Proc. of ESORICS, LNCS 3193, pp. 407-422, Springer, 2004.

[3] R. Sommer, A. Feldmann: NetFlow: Information Loss or Win? In: Proc. of the 2nd Internet Measurement Workshop, Marseille, France, 2002.

[4] C. Kruegel, T. Toth: Using Decision Trees to Improve Signature-based Intrusion Detection In: Proc. of RAID 2003, LNCS 2820, pp. 173-191, Springer, 2003.

[5] K. Anagnostakis, E. Markatos, S. Antonatos, M. Polychronakis: E2xB: A domainspecific string matching algorithm for intrusion detection. In: Proc. of IFIP SEC 2003.

[6] M. Meier: A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In: Proc. of 7th ISC 2004, LNCS 3225, pp. 158–169, Springer, 2004.

[7] U. Flegel, M. Meier: Towards a Scalable Approach to Tailoring the Disclosure of Pseudonymous Audit Data to Misuse Detection Signatures. Internes Diskussionspapier, 2002.

[8] S. Schmerl: Entwurf und Entwicklung einer effizienten Analyseeinheit für Intrusion-Detection-Systeme. Diplomarbeit, Lehrstuhl Rechnernetze, BTU Cottbus, 2004.

[9] G. Vigna, S.T. Eckmann, R.A. Kemmerer: The STAT Tool Suite. In: Proc. of DISCEX 2000, pp. 46-55, IEEE Computer Society Press, 2000.

[10] S. Kumar: Classification and Detection of Computer Intrusions. PhD thesis, Dept. of Computer Science, Purdue University, West Lafayette, IN, 1995.

[11] S.T. Eckmann, G. Vigna, R.A. Kemmerer: STATL: An Attack Language for State-based Intrusion Detection. In: Journal of Computer Security, vol. 10, no 1/2, pp. 71-104, 2002.

[12] F. Puppe: Einführung in Expertensysteme, Springer-Verlag, Berlin, 1991.

[13] P.G. Neumann, A.P. Porras: Experience with EMERALD to Date. In: Proc. of 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, 1999.

[14] P.E. Proctor: Audit reduction and misuse detection in heterogeneous environments: Framework and application. In: Proc. of 10th ACSAC, 1994, pp. 117-125.

[15] M. Sobirey, B. Richter; H. König: The Intrusion Detection System AID. In: Proc. of IFIP TC6 / TC11 conference, pp. 278 – 290, Chapman & Hall, London, 1996.

[16] U. Lindqvist; P.A. Porras: Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). In: Proc. of IEEE Symposium on Security and Privacy, pp. 146-161, IEEE Computer Society Press, California, 1999.

[17] G. Riley: CLIPS – A Tool for Building Expert Systems. May 2004, http:// www.ghg.net/clips/CLIPS.html

[18] Talarian Corporation: RTie Inference Engine. In: Talarian Corporation: RTworks 3.5. Mountain View, CA, 1995.

[19] R. Krauz: Implementierung eines auf dem Expertensystem-Tool CLIPS basierenden Intrusion Detection Systems. Studienarbeit, Lehrstuhl Rechnernetze, BTU Cottbus, 2004.

[20] C. L. Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: Artificial Intelligence, vol. 19, no. 10, pp. 17-37, 1982.

[21] A. V. Aho, R. Sethi, J. D. Ullman: Compilers - Principles, Techniques and Tools. Addison-Wesley, 1988.