Intrusionserkennung mit System Calls und Hidden Markov Models

Thorsten Knoll

Studiengang Angewandte Informatik, Hochschule RheinMain
thorsten.knoll@student.hs-rm.de

Zusammenfassung

Hidden Markov Models (HMMs) haben sich in der Sprach- und Texterkennung sowie in der Gen-Sequenz-Analyse in den letzten Jahrzehnten etabliert. Die statistische Mustererkennung mit HMMs ist vor Allem für zeitlich organisierte Daten wirksam anzuwenden. Obwohl HMMs nicht einfach zu modellieren und trainieren sind, bieten sie Vorteile gegenüber anderen Verfahren. So zum Beispiel bei der Erkennung komplexer Muster, die aus einfacheren Komponenten zusammengesetzt sind (Schrift oder Sprache) [1]. Dieser Artikel diskutiert die Grundlagen von HMMs und geht spezieller auf eine Anwendung im Bereich der Software-Intrusionserkennung ein [2].

I. Hidden Markov Models

I.1 Die Idee

Mit HMMs werden beobachtbare Daten und die darin versteckten Informationen in einem Modell mit Zustandsautomaten, Übergangswahrscheinlichkeiten und Emissionen abgebildet (Figure 1). Die Emissionen sind die beobachtbaren Daten und der Zustandsautomat stellt die versteckten Informationen dar.
Zur Erstellung von HMMs sind Trainingsdaten notwendig, in welchen sowohl die Emissionen als auch die Informationen bekannt sind. Das Ziel von HMMs ist die Erkennung der zugrunde liegenden Informationen in weiteren, beobachtbaren Daten. Stelle man sich als Beispiel die Erkennung von Sprache vor. Die Trainingsdaten sind Sprachaufzeichnungen mit Ihrem enthaltenen semantischen Inhalt.

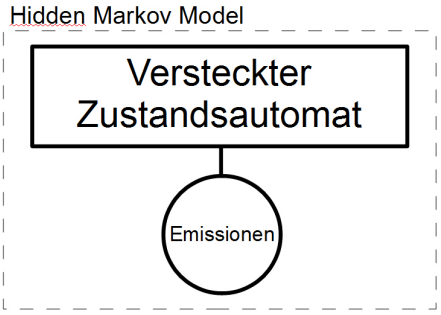

Abbildung 1: Hidden Markov Model Schema

Und erkannt werden soll der semantische Inhalt von live eingesprochener Sprache. Anwendungen solcher Spracherkennungssysteme sind mittlerweile weit verbreitet (Navigation im Auto). Ein weiteres Beispiel ist die automatisierte Texterkennung (OCR). Als Trainingsdaten sind Bilder von Texten mit ihrem syntaktischen Inhalt bekannt und es soll die Syntax von weiteren Textbildern erkannt werden.

Beispiele:

Emissionen	**Kodierte Informationen**
Münzwürfe	Faire Münze?
Sprache	Semantik
Texte	Syntax

I.2 Zustandsautomaten und Emissionen

Die Grundlage von HMMs ist die Abbildung von natürlichen Vorgängen mit Hilfe von Zustandsautomaten. Hierfür wird davon ausgegangen, das diese natürlichen Vorgänge statis-

tisch abbildbar und in eine zeitliche Abfolge zu bringen sind. Sind die Daten diskreter Natur, wie zum Beispiel bei Münzwürfen, ist die zeitliche Einteilung vorgegeben. Kontinuierliche Daten (Sprache) werden quantisiert und so in eine diskrete Abfolge gewandelt. Das Standardverfahren ist hierbei die Vektorquantisierung, wird aber in diesem Artikel nicht weiter besprochen [1].
Um den Zustandsautomaten eines HMMs zu erstellen werden folgende Annahmen getroffen:

1. Die Anzahl der Zustände N ist endlich.
2. Die Zustände sind Zeitpunkten zugeordnet und besitzen Übergangswahrscheinlichkeiten. Das Gedächtnis des Automaten ist genau einen Zustand groß (Markov-Eigenschaft).
3. Nach jedem Übergang ist eine Emission zu beobachten, die mit Ihrer Wahrscheinlichkeit an den erreichten Zustand gekoppelt ist.

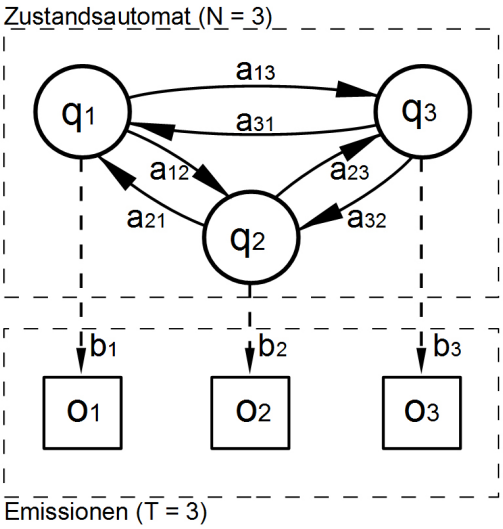

Abbildung 2: Beispiel eines HMMs mit 3 Zuständen

In Figure 2 ist ein einfaches HMM abgebildet. q_1, q_2 und q_3 sind die versteckten Zustände, a_{12}...a_{31} die Übergangswahrscheinlichkeiten, b_1,b_2 und b_3 die Emissionswahrscheinlichkeiten und O_1,O_2 und O_3 die beobachtbaren Emissionen.
Hieraus ergibt sich eine formale Definition des HMMs [3]:

Parameter	Beschreibung
T	Anzahl der Zeitpunkte
N	Anzahl der Zustände
M	Anzahl der Emissionsmöglichkeiten
Q	Zustandsmenge $\{q_1, q_2, ..., q_N\}$
V	Menge aller möglichen Emissionen $\{v_1, v_2, ..., v_M\}$
$A = \{a_{ij}\}$	Verteilung der Übergangswahrscheinlichkeiten mit $a_{ij} = P(q_j(t+1) \mid q_i(t))$ und $1 \leq i, j \leq N$
$B = \{b_i(k)\}$	Verteilung der Emissionswahrscheinlichkeiten mit $b_i(k) = P(v_k(t) \mid q_i(t))$, $1 \leq k \leq M$ und $1 \leq i \leq N$
$\pi = \{\pi_i\}$	Verteilung der Startzustandswahrscheinlichkeiten mit $\pi_i = P(q_i(t=1))$ und $1 \leq i \leq N$

HMMs können als 5-Tupel in der Kurzschreibweise $\lambda = (Q, V, A, B, \pi)$ notiert werden. Auch die weitere Verkürzung auf ein 3-Tupel $\lambda = (A, B, \pi)$ ist eine gebräuchliche Notation, meistens wenn nur die Verteilungen von Interesse sind.

I.3 Wahrscheinlichkeit einer Emissionssequenz

Bevor die Optimierung der HMM-Parameter besprochen wird, ist zum allgemeinen Verständnis der Problemstellung einen Blick auf die Berechnung der folgenden Wahrscheinlichkeit hilfreich. Sei O eine Sequenz von Emissionen $o_1, o_2, ..., o_T$. Dann ist die Wahrscheinlichkeit von O bei gegebenem HMM λ:
$P(O \mid \lambda)$.

Diese zu berechnende Wahrscheinlichkeit wird auch Produktionswahrscheinlichkeit genannt und ist für die Erstellung, Optimierung und Benutzung von HMMs von entscheidender Bedeutung. Das zu lösende Problem der Produktionswahrscheinlichkeit ist als Evaluierungsproblems bekannt [1]. Zum Verständnis des Evaluierungsproblems werden zwei Lösungswege gezeigt. Der zweite Lösungsweg liegt in einer effizienteren Aufwandsklasse und baut auf den Erkenntnissen des ersten Lösungswegs auf.

I.3.1 Aufwändige Methode

Da alle notwendigen Eingabeparameter vorhanden sind, kann $P(O \mid \lambda)$ in einer Art 'brute force'-Methode berechnet werden. Hierzu werden alle möglichen Zustandsfolgen aus der Menge der Zustände Q ermittelt. Sei Z die Menge dieser Zustandsfolgen $z_1, z_2, ..., z_T$. Da jede Emission genau einem Zustand zugehörig ist, folgt $\#O = \#Z = T$ (siehe I.2(3)). Von jeder dieser möglichen Zustandsfolgen wird die Wahrscheinlichkeit berechnet und zum Ergebnis aufsummiert. Für die Produktionswahrscheinlichkeit ergibt sich [3], [1]:

$$P(O \mid \lambda) = \sum_{\vec{z} \in Z} \sum_{x=1}^{T} P(O \mid z_x, \lambda) P(z_x \mid \lambda)$$

Formuliert man diese Berechnung als Algorithmus, ergibt sich dafür eine Aufwandsklasse $O(TN^T)$, also exponentiell zur Anzahl der Zeitpunkte (Länge der Emissionssequenz) [1].

I.3.2 Forward-Algorithmus

Um die Produktionswahrscheinlichkeit in einer besseren Aufwandsklasse algorithmisch berechnen zu können, macht man sich die Markov-Eigenschaft zu nutze (siehe I.2(2)). Da der Automat sich nur genau einen Zustand merken kann, müssen zu einem Zeitpunkt $t+1$ nur die möglichen Zustandsvorgänger zum Zeitpunkt t betrachtet werden. Diese Eigenschaft lässt sich für einen rekursiven Algorithmus benutzen [1]. Für Emissionssequenzen die länger als die Anzahl der Zustände sind ($\#O > \#Q$) ergibt sich noch ein positiver Effekt: Der Schubfachschluss, auch Taubenschlagprinzip genannt. Wird ein Zustand ein zweites Mal besucht, sind die möglichen Vorgänger nach dem oben genannten Vorraussetzungen genau die selben wie beim ersten Besuch [3].
Mit festem HMM λ, dem Zeitpunkt t, der Emissionsfolge von o_1 bis o_t und dem erreichten Zustand z_t kann die Forward-Variable definiert werden [3], [1]:

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, z_{t=i} \mid \lambda)$$

Es gilt bei Induktion über diese Forward-Variable:
1. Induktionsstart:

$$\alpha_1(i) = \pi_i b_i(o_1), i \leq N$$

2. Induktionsschritt:
für alle $1 \leq t \leq T-1$ und $1 \leq j \leq N$ gilt

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_i(o_{t+1})$$

3. Nun lässt dich die Produktionswahrscheinlichkeit berechnen:

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Die Erwartung einer besseren Aufwandsklasse als in der zuerst vorgestellten Lösung für die Produktionswahrscheinlichkeit wird durch die wesentlich geringere Anzahl an Multiplikationen erfüllt. In Schritt 2 wird $\alpha_{t+1}(j)$ in $O(N)$ berechnet. Und danach in Schritt 3 davon die Summe von 1 bis N gebildet. Für T Zeitschritte ergibt das die Aufwandsklasse $O(N^2T)$ und damit wesentlich weniger Aufwand als die Methode in I.3.1. Dieser Forward-Algorithmus wird auch aktuell für HMMs verwendet.
An dieser Stelle sei auch noch der zugehörige Backward-Algorithmus erwähnt. In

gleicher Weise der Forward-Definition wird eine Backward-Variable definiert und mit Induktion zu dieser Formel gewandelt:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j))$$

mit $1 \leq i, j \leq N$.
Auf diese beiden Algorithmen wird im folgenden Kapitel weiter Bezug genommen. Zusammen ergeben sie den Forward-Backward-Algorithmus.

I.4 Optimierung

Nach der Definition der HMMs und der rekursiven Berechnung der Produktionswahrscheinlichkeiten stellt sich die Frage nach einem Algorithmus zur Erstellung optimal auf die Trainingsdaten abgestimmter HMMs. Ein solcher Algorithmus zur Erstellung von HMMs ist nicht bekannt [1]. Soll ein neuer HMM erstellt werden, muss das Grundmodell entweder selbst entwickelt oder ein bestehendes Modell modifiziert werden. Dieses Grundmodell kann dann optimiert werden. Hier werden verschiedene Ansätze zur Optimierung besprochen.

I.4.1 Gedrehte Produktionswahrscheinlichkeit

Dreht man die Idee der Produktionswahrscheinlichkeit herum, erhält man die Frage nach der Wahrscheinlichkeit des HMM λ für die Emissionssequenz O. So können mehrere HMMs λ_1, λ_2,... verglichen werden, die dieselbe Emissionsfolge abbilden. Diese ist mit Bayes einfach zu formulieren [1]:

$$P(\lambda_j \mid O) = max \frac{P(O \mid \lambda_i) P(\lambda_i)}{P(O)}$$

$P(O)$ ist nicht von λ_i abhängig und wirkt sich daher nicht auf das Maximum aus. In der Praxis wird nach [1] auch meist noch die Wahrscheinlichkeit des Models $P(\lambda_i)$ vernachlässigt weil diese vorab bekannt sein muss(A-priori). Zur Bewertung von HMMs mit gleicher Emissionen werden demnach meistens doch nur die Produktionswahrscheinlichkeiten $P(O \mid \lambda_i)$ betrachtet, deren Berechnung weiter oben vorgestellt wurde.

I.4.2 Die optimale Zustandsfolge

Eine andere Frage im Zusammenhang mit der Produktionswahrscheinlichkeit ist die nach der optimalen Zustandsabfolge z^*. Dabei werden die Emissionsfolge O und das HMM als gegeben betrachtet. Das ermitteln von z^* wird als Dekodierung oder Aufdeckung der internen Vorgänge eines HMM bezeichnet [1]. Das Problem lässt sich also wie folgend formulieren:

$$z^* = \underset{z}{\operatorname{argmax}} P(z \mid O, \lambda)$$

Wie in der Berechnung der Produktionswahrscheinlichkeit ist auch hier der erste Lösungsansatz eine 'brute-force'-Methode in welcher die Wahrscheinlichkeiten aller möglichen Zustandsfolgen betrachtet werden und die wahrscheinlichste davon als z^* genommen wird. Da auch hier wieder mit der Aufwandsklasse $O(TN^T)$ berechnet wird, ist dieses Verfahren nicht praktikabel [1].

I.4.3 Der Viterbi-Algorithmus

Es gibt mit dem Viterbi-Algorithmus, welcher dem Forward-Backward-Algorithmus sehr ähnlich ist, eine Möglichkeit der Berechnung in $O(\mid z \mid^2 T)$ [4]. Wobei $\mid z \mid$ die Länge der Zustandsfolge z ist. Im Viterbi-Algorithmus wird wieder die Markov-Eigenschaft in Verbindung mit einer Induktion angewandt. Der Unterschied besteht in einer Maximierung des bisherig berechneten Zustandspfads. Hierdurch entsteht eine Maximum-Likelihood-Zustandsfolge [4]. Zum Anfang wird wieder eine Variable definiert, diesmal mit der genannten Maximierung [3], [1]:

$$\delta_t(i) = \underset{z_{1:t-1}}{\operatorname{argmax}} P(o_1, o_2, ..., o_{t-1}, z_{t=i} \mid \lambda)$$

1. Induktionsstart:

$$\delta_1(i) = \pi_i b_i(o_i), i \leq N$$

2. Induktionsschritt:
für alle $1 \leq t \leq T-1$ und $1 \leq j \leq N$ gilt

$$\delta_{t+1}(j) = \max_i [\delta_t(i) a_{ij}] b_j(o_{t+1})$$

3. So lässt sich die maximierte Produktionswahrscheinlichkeit P^* berechnen:

$$P^*(O \mid \lambda) = P(O, z^* \mid \lambda) = \max_i \delta_T(i)$$

Da $\delta_T(i)$ aber nur den Abschlusszustand z^* maximiert, müssen die besuchten Zustände ausgehend vom Ende der Zustandsfolge noch ermittelt werden. Hierzu bietet sich die in I.3.2 besprochene Backward-Berechnung an. Eine weitere Variable wird hierzu mit Ihrer Maximierung definiert und per Induktion entwickelt [1]:

$$\psi_t(j) = \operatorname*{argmax}_i \delta_{t-1}(i) a_{ij}$$

1. Induktionsstart:

$$\psi_1(i) = 0, i \leq N$$

2. Induktionsschritt:
für alle $1 \leq t \leq T-1$ und $1 \leq j \leq N$ gilt

$$\psi_{t+1}(j) = \operatorname*{argmax}_i \delta_t(i) a_{ij}$$

3. Und daraus folgt:

$$z_T^* = \operatorname*{argmax}_j \delta_T(j)$$

Nimmt man beide Ergebnisse für die Forward- und Backward-Berechnung zusammen, lässt sich die optimale Zustandsfolge dann rekursiv wie folgend berechnen:

$$z_t^* = \psi_{t+1} z_{t+1}^*$$

I.4.4 λ-Parameteroptimierung

Wie weiter oben beschrieben gibt es keinen Algorithmus zur Erstellung von HMMs. Für die Optimierung der HMM-Parameter A,B und π sind verschiedene Verfahren bekannt. In [3] werden hier als hauptsächlich verwendete Verfahren die iterative Baum-Welsch-Methode und das Gradienten-verfahren genannt. Für eine sinnvolle Optimierung des Modells werden im Folgenden einige Annahmen getroffen:

1. Es muss eine Größe als Modellgüte definiert werden. Bisher wurde die Produktionswahrscheinlichkeit vorgestellt, die als Kandidaten dafür in Frage kommen.
2. Das Optimierungsverfahren soll die Produktionswahrscheinlichkeit ausschließlich verbessern. Es wird damit die Abbruchbedingung definiert, dass der Algorithmus stoppt wenn keine weitere Verbesserung im letzten Schritt erreicht wurde.
3. Aus der Beschaffenheit von HMMs ergibt sich, dass nur mit Erwartungsgrößen gerechnet werden kann, nicht mit absoluten Werten.
4. Die optimierten HMM-Parameter sind als $\hat{a}_{ij}$, $\hat{b}_j(o_k)$ und $\hat{\pi}_i$ definiert. Diese stellen das Ergebnis der Optimierung dar.

Es wird nun der Baum-Welch-Algorithmus vorgestellt. Er verwendet als Größe für die Modellgüte die Gesamtproduktionswahrscheinlichkeit $P(O \mid \lambda)$. Diese Wahrscheinlichkeit darf also nach obigen Annahmen durch die iterative Anwendung des Algorithmus nicht kleiner werden, bis dann zum Abbruch ein potentiell lokales Maximum erreicht ist.
Da in der Produktionswahrscheinlichkeit alle möglichen Pfade betrachtet werden, ist die Definition einer Übergangswahrscheinlichkeit zwischen den Zuständen z_i und z_j zu einem Zeitpunkt t notwendig:

$$\gamma_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$

Die Zusammensetzung der Übergangswahrscheinlichkeit ist in Figure 3 übersichtlich dargestellt. Summiert man $\gamma_t(i,j)$ über alle Zustandsnachfolger für j, ergibt sich die Wahrscheinlichkeit für einen einzelnen Zustand z_i

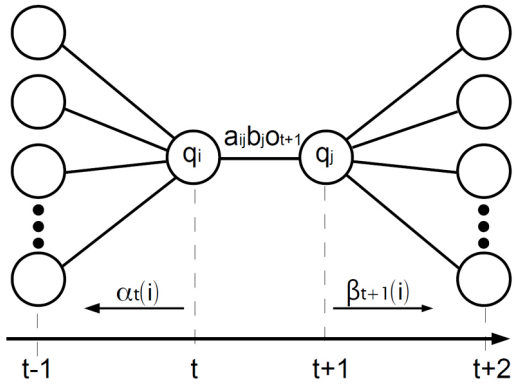

Abbildung 3: Schema zur Übergangswahrscheinlichkeit

im Zeitpunkt t:

$$\gamma_t(i) = \sum_{j=1}^{N} \gamma_t(i,j)$$

Mit diesen zwei Wahrscheinlichkeiten $\gamma_t(i,j)$ und $\gamma_t(i)$ ergeben sich aus einer hier nicht weiter betrachteten Herleitung folgende Berechnungen für die gesuchten, optimierten Parameter:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\hat{b}_j(o_k) = \frac{\sum_{t:o_t=o_k}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\hat{\pi}_i = \gamma_1(i)$$

Damit ist ein mögliches Verfahren zur Optimierung von HMMs, das Baum-Welch-Verfahren, vorgestellt. Es gibt noch weitere und ausführlichere Verfahren auf die hier nicht eingegangen wird.

II. Erkennung von Intrusionen mit HMMs

HMMs sind ein komplexes aber auch mächtiges Werkzeug im Bereich des maschinellen Lernens. Im folgenden wird die Wirksamkeit von HMMs bei der Erkennung von Intrusionen anhand von System Calls auf Unix basierten Systemen betrachtet. Zum Vergleich werden mehrere verschiedene Erkennungsmodelle ausgewertet. Die vorgestellten Ergebnisse beruhen allesamt auf dem Artikel 'Detecting Intrusions using System Calls: Alternative Data Models' [2].

II.1 System Calls

Programme erzeugen während Ihrer Laufzeit Sequenzen von System Calls. Um mit diesen Sequenzen Intrusionen zu erkennen, werden Abweichungen vom Standardverhalten gesucht. Diese Abweichungen erlauben dann Aussagen über die Wahrscheinlichkeit einer Intrusion. Hierzu werden die live erzeugten System Call Sequenzen mit Hilfe von statistischen Modellen auf Basis der vorhandenen Daten ausgewertet.

II.2 Erkennungsmodelle

Die verwendeten Modelle für den Vergleich sind:

1. Enumeration von Sequenzen (stide):
Es werden zusammenhängende Sequenzen fester Länge aus den vorhandenen Daten generiert und als normales Verhalten definiert. Diese Sequenzen werden in einem Baum gespeichert. Während der Laufzeit eines Programms werden dann die auftretenden Sequenzen im Baum gesucht. Das Vorhandensein einer Sequenz im Suchbaum stellt dann die Einteilung in normales oder intrusives Verhalten dar. Diese Methode ist als 'sequence time-delay embedding' (stide) bekannt.

2. Frequenzbasiertes Modell (t-stide):
Hier werden Häufigkeitsverteilungen von System Call Mustern in den Sequenzen analysiert. Im verwendeten Modell werden auftretende, kurzzeitige Verteilungen mit den erlernten, langfristigen Verteilungen

verglichen. Die langfristigen Verteilungen erlernen dabei während der Ausführung das normale Verhalten.

3. Data Mining (RIPPER):
'Repeated Incremental Pruning to Produce Error Reduction' (RIPPER) erlernt Regeln zur Erkennung von System Call Sequenzen. Für die Verletzung dieser Regeln durch die live auftretenden Sequenzen wird ein Bewertungsschema erstellt. Mehrfache Verletzung der Regeln weist dabei auf Intrusion hin.

4. HMMs:
Für die verwendeten HMMs wurden Zustandsautomaten mit 20-60 Zuständen gewählt, wobei die Zustände vollständig verbunden sind. Es ergeben sich daraus vollständig besetzte Matrizen für die Übergangswahrscheinlichkeiten (A) und die Emissionswahrscheinlichkeiten (B). A und B wurden zufällig initiiert und dann mit dem oben besprochenen Baum-Welsch-Algorithmus optimiert. Im Gegensatz zu den vorherigen Modellen prüfen HMMs bei jeder Emission (und damit auch jedem Zustandsübergang) die Wahrscheinlichkeit für eine Intrusion. Auch ist zu bemerken das die Trainigszeit für die HMMs wesentlich länger war, als für die anderen 3 Modelle. Während die stide, t-stide und RIPPER in den meisten Fällen nach einigen Stunden trainiert waren, haben die HMMs bis zu 2 Monate benötigt.

II.3 Bewertungskriterien für die Erkennungsmodelle

Um die Erkennungsmodelle vergleichen zu können, müssen Bewertungskriterien definiert werden. Sei die Erkennung von Intrusionen positiv, ergeben sich die folgenden Kriterien:

1. Richtig oder falsch positiv:
Als richtig positiv wird die korrekte Erkennung einer Intrusion definiert. Demnach ist falsch positiv die fehlerhafte Erkennung einer Intrusion.
2. Richtig oder falsch negativ:
Als richtig negativ wird die korrekte Erkennung normalen Verhaltens definiert. Demnach ist falsch negativ die fehlerhafte Erkennung normalen Verhaltens.

Das ideale Modell hätte keine falsch positiven und keine falsch negativen Erkennungen. Da die Modelle Wahrscheinlichkeiten abbilden, muss noch eine Grenzschwelle (Threshold) für richtig oder falsch definiert werden.
Für die Bewertung und den Vergleich der Erkennungsmodelle wurden die Threshold-Werte so gewählt, das eine richtig positiv Maxmimierung auf mindestens 95% Erkennungsrate erreicht wurde.

II.4 Testdaten

Für den Vergleich der Erkennungsmodelle wurden System Call Daten der folgenden Programme verwendet:
lpr, named, xlock, login, ps, inetd, stide und sendmail.
Die Intrusionsszenarien waren hierbei:
Buffer Overflow, Symbolic Link Attacks, Trojaner und Denial-of-Service Attacken.
Die Daten enthalten die Intrusions-Szenarien gesondert, so dass das Training und Erkennung getrennt vorgenommen werden konnten. Ausserdem lagen für das Programm lpr zwei unterschiedliche Datensätze von verschiedenen Universitäten (MIT und UNM) vor.

Programm	**Intrusionen**	**System Calls**
lpr MIT	1.001	2.926.304
lpr UNM	1.001	2.027.468
named	2	9.230.572
xlock	2	16.937.816
login	9	8.894
ps	26	6.144
inetd	31	541
stide	105	15.618.237
sendmail	-	44.500.219

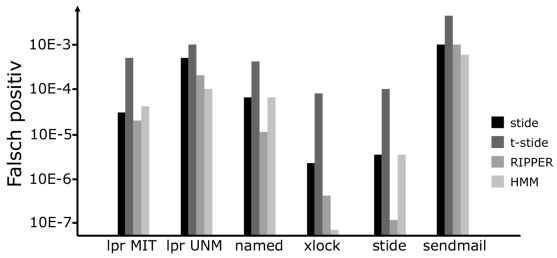

Abbildung 4: Falsch positiv Erkennungsraten

II.5 Ergebnis des Vergleichs

Wie in Figure 4 zu sehen, sind die Ergebnisse der Erkennungsraten für die verschiedenen Programme sehr unterschiedlich. Es wurden für die Auswertung einige Restriktionen vorgenommen. So sind die Längen der Sequenzen recht kurz auf 6 und 10 System Calls beschränkt worden. Ausserdem wurde die Bewertung auf die falsch positiv Erkennungen fokusiert, bei den weiter oben beschriebenen Treshold-Annahmen. Die Daten der Programme login, ps und inetd sind in der Grafik nicht enthalten, da die Menge der Testdaten und die Anzahl der unterschiedlichen System Call Sequenzen kein relevantes Ergebniss ermöglich haben.
Es wird in [2] deutlich, das HMMs zwar gute Ergebnisse für die Intrusionserkennung mit System Calls liefern, aber die Auswahl des Erkennungsmodells alleine nicht ausreicht um einen Favoriten auszumachen. Vielmehr hängt die Auswahl des Erkennungsmodells mit der konkreten Problemstellung zusammen. Keines der bewerteten Erkennungsmodelle bietet alleinstehend gute Ergebnisse für alle ausgesuchten Programme und deren System Call Sequenzen. Es ist aber zu bemerken das die frequenzbasierte Erkennung mit t-stide überdurchschnittlich schlechte Erkennungen liefert. Beachtet man weiterhin die benötigte, hohe Rechenleistung von HMMs im Vergleich zu dem einfacheren Emnumerationsverfahren stide stellt sich stide als das zu favorisierende Erkennungsmodell mit den gewählten Testdaten dar. In [2] wird als möglicher Grund für dieses Ergebnis genannt, das System Call Daten scheinbar regulär genug sind um mit einfachen Erkennungsmodellen gut abgebildet zu werden.

Literatur

[1] Gernot A. Fink. *Mustererkennung mit Markov-Modellen : Theorie, Praxis, Anwendungsgebiete*. PhD thesis, Stuttgart [u.a.], 2003. Zugl.:Bielefeld, Univ., Habil.-Schr., 2003.

[2] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145, 1999.

[3] L. Rabiner and B.H. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, Jan 1986.

[4] Shaunak Chatterjee and Stuart Russell. A temporally abstracted viterbi algorithm. In *Uncertainty in Artificial Intelligence (UAI)*, pages 96–104. AUAI Press, 2011.