

Zustandsbasierte Anomalie-Erkennung im HTTP Protokoll

Benedikt Vamos, B.Sc.

Fachhochschule Oberösterreich, Campus Hagenberg,
Studiengang Sichere Informationssysteme
A-4232 Hagenberg, Österreich
Email: benedikt.vamos@fh-hagenberg.at

Zusammenfassung - Anomalie-Erkennungssysteme, herkömmliche Web Application Firewalls und Intrusion Detection Systems überprüfen einzelne HTTP Transaktionen auf Gültigkeit. Die Reihenfolge der HTTP Transaktionen und anderer zustandsbasierter Abläufe im System bleibt dabei unberücksichtigt. Beschriebene zeitliche Abläufe stellen gleichfalls interessante Informationen für die Anomalie-Erkennung dar. Diese Publikation beschäftigt sich mit zeitlichen Abläufen und wie diese für die zustandsbasierte Anomalie-Erkennung genutzt werden können. Zusätzlich werden die Implementierung eines Prototyps und Testergebnisse präsentiert.

I. Bekannte Ansätze, Erweiterung und Abgrenzung

Das Internet bietet Angreifern eine steigende Bandbreite an Angriffsmöglichkeiten auf Unternehmen, Behörden, Vereine und private Personen. Immer mehr Systeme und Anwendungen sind durch eine Web-Oberfläche mit dem Internet verbunden. Auch die Anzahl der Web-Auftritte, Plattformen und Geschäftsmodelle, die sich das Internet zu Nutze machen, steigt. Web-Anwendungen sind für Angreifer lohnende Ziele, da sie oft mit sensiblen Daten arbeiten und frei zugänglich sind. Mitunter trägt eine Web-Anwendung auch zum Ruf eines Unternehmens bei. In Folge dessen stellt sich die Frage, wie Web-Applikationen besser gegen Angriffe geschützt werden können. Viele Angriffe richten sich gegen die Anwendungsebene dieser Systeme. Eingerichtete Sicherheitsmechanismen sind oft nicht in der Lage, auf Angriffe dieser Ebene zu reagieren, da herkömmliche signaturbasierte Web Application Firewalls (WAF) und Intrusion Detection Systems (IDS) bei unbekanntem Angriffsverfahren an ihre Grenzen stoßen und die zunehmende Komplexität von Informationssystemen die Anforderungen an diese Sicherheits-einrichtungen erhöht.

Signaturbasierte Erkennungssysteme sind weit verbreitet und es gibt eine Vielzahl an Produkten in diesem Bereich. Zwei der bekanntesten Open Source Web Application

Firewalls sind ModSecurity und Snort. ModSecurity ist ein Apache-Modul, welches in den Web-Server integriert wird. ModSecurity ermöglicht Monitoring, Logging und Echtzeitanalysen [Mods11]. Snort ist ein Network Intrusion Detection/Prevention System. Es ermöglicht neben der Analyse des HTTP Protokolls auch die Auswertung anderer Protokolle wie zum Beispiel FTP, SSH, DNS und ARP. Es ist auf allen Ebenen des OSI-Modells, außer Layer 1, einsetzbar [Snor11]. Eine Alternative zu diesen sogenannten „*misuse detection*“-Systemen stellt die Anomalie-Erkennung (auch „*outlier detection*“ genannt) dar. Bei der Anomalie-Erkennung wird das Soll-Verhalten der Benutzer in einem System oder des Systems selbst anstelle des Fehlverhaltens gelernt. Dieser Ansatz erlaubt das Erkennen von anormalen Verhaltensmustern wie z.B. Angriffen, ohne bisher von einer Schwachstelle im System gewusst zu haben [DeDo86].

Auf dem Gebiet der Anomalie-Erkennung gibt es erst wenige Referenzsysteme. Mit dieser Thematik beschäftigt sich Kirchner in [Kirc10]. In [Kirc10] wird die Umsetzung einer Anomalie-Erkennung im HTTP Protokoll mit Hilfe des instanzbasierten Lernens und der KNN-Klassifikation beschrieben. Es werden keine Signaturen für die Anomalie-Erkennung verwendet. Stattdessen wird für jede einzelne URL im System gelernt welche Ausprägung diese hat. Nachdem eine bestimmte Anzahl an HTTP Transaktionen pro URL gelernt ist, werden zukünftige Transaktionen mit dem Erlernten verglichen. In der sogenannten „*Lernphase*“ wird ausgewertet, welche HTTP Transaktionen es in der zu beobachtenden Web-Applikation gibt und welche Eigenschaften diese aufweisen. Die Auswertung der einzelnen Transaktionen wird in Form eines Vektors gespeichert. In der „*Betriebsphase*“ werden neue Transaktionen von Benutzern mit denen der Lernphase verglichen und berechnet, ob diese den gelernten ähnlich sind. Weicht eine neue Transaktion stark von den gelernten ab, wird von einer Anomalie ausgegangen. Dieser Ansatz liefert im Betrieb Informationen über die Gültigkeit einzelner HTTP Transaktionen. Eine HTTP Transaktion ist gültig, wenn in der Lernphase ähnliche Transaktionen

aufgezeichnet wurden. Dabei wird nicht ausgewertet, an welcher Stelle oder in welchem Zustand eines Systems diese Transaktionen vorkommen. In einem System ist es beispielsweise der Fall, dass das Ändern oder Abrufen von Datenbankinhalten erst nach einer Anmeldung erlaubt ist. Der Ansatz von Kirchner, aber auch herkömmliche Web Application Firewalls und Intrusion Detection Systems, überprüfen den Login-Request und den Request zum Abfragen oder Löschen der Datenbankinhalten auf seine Gültigkeit. Die Reihenfolge der HTTP Transaktionen und andere zustandsbasierte Abläufe im System bleiben dabei unberücksichtigt. Die beschriebenen zeitlichen Abläufe liefern gleichfalls interessante Informationen für die Anomalie-Erkennung. An diesem Punkt setzt die zustandsbasierte Anomalie-Erkennung an. Ihre Aufgabe ist es, Zustände in einem System zu lernen, zu erkennen und die im System üblichen Beziehungen zwischen ihnen abzubilden. Zustandsbasierte Anomalie-Erkennung eignet sich besonders als Erweiterung zur Anomalie-Erkennung, aber auch erweiternd zur signaturbasierten Erkennung.

Die Aufgaben der zustandsbasierten Anomalie-Erkennung liegen nicht in der Auswertung der Payload oder anderer übertragener Daten einzelner HTTP Transaktionen. Stattdessen werden die Attribute der Transaktionen genutzt, um Zustände und zeitliche Abläufe zu beschreiben. Es ist beispielsweise nicht relevant, ob in einem Formularfeld ein (un)erlaubter Text eingegeben wird, sondern lediglich die Tatsache, dass üblicherweise an dieser Stelle der Webseite eine Texteingabe erfolgt. Die Benutzer einer Web-Applikation lösen mit ihrem Verhalten bestimmte Zustandswechsel im System aus. In diesen Zustandswechseln spiegelt sich das Verhalten der Benutzer wider und es ist möglich, dieses mit einem vorhandenen, zuvor berechneten Markov Modell zu vergleichen um darauf zu schließen, ob es sich um typisches Verhalten handelt oder nicht.

Wie diese Publikation beschäftigen sich auch Ariu, Giacinto und Predisci [ArGP07] mit dieser Thematik. In deren Publikation [ArGP07] wird die Verwendung des Hidden Markov Modells zur Anomalie-Erkennung anhand des FTP Protokolls beschrieben. Bei HTTP handelt es sich allerdings um ein komplexeres Protokoll, wobei sich ein Request aus mehreren Parametern zusammensetzt und andere Eigenheiten zu beachten sind.

II. Markov Modelle

Bei dem Markov Modell handelt es sich um einen stochastischen Prozess. Ein stochastischer Prozess beschreibt zeitliche Vorgänge, die zufällige Schwankungen aufweisen [Sieg05] [Unip11]. Die Markov Ketten (Sequenzen), welche im Markov Modell abgebildet werden, werden mit diskreten Zeitparametern betrachtet. Zumeist ist auch der Zustandsraum diskret. In Markov Ketten ist ein Zustand zum Zeitpunkt m lediglich durch die Werte im Zustand m sowie dem vorangegangenen Zustand $m-1$ abhängig. Der Zustand ist von früheren Werten unabhängig [Fink03, S.74]. Eine Markov Kette besteht aus einer Reihe von Elementen und wird durch die

Anfangswahrscheinlichkeit (π) des ersten Elements, der Zustandsreihenfolge und den Übergangswahrscheinlichkeiten beschrieben. Die Übergangswahrscheinlichkeit ist die Wahrscheinlichkeit, mit der von einem bestimmten Zustand in einen anderen bestimmten Zustand gesprungen wird. Für jeden möglichen Übergang zwischen Zuständen gibt es eine Übergangswahrscheinlichkeit. Je nach Markov Modell existieren auch Übergangswahrscheinlichkeiten von „0“ zwischen manchen Zuständen.

Das Hidden Markov Modell (HMM) stellt eine Erweiterung des herkömmlichen Markov Modells dar. Beim Hidden Markov Modell wird davon ausgegangen, dass der Beobachter nicht sehen kann, in welchem Zustand sich ein System befindet. Lediglich durch Emissionen kann mit einer bestimmten Wahrscheinlichkeit darauf geschlossen werden, in welchem Zustand sich das System zum Zeitpunkt der Messung befindet. Emissionen können auch auf Zustandsübergänge angewandt werden und damit die Wahrscheinlichkeit eines bestimmten Zustandsübergangs ausdrücken. Bei einer Beobachtung zum Zeitpunkt m werden die Emissionen des Systems gemessen und daraus ein Sample erzeugt. Diese Samples (i.e. Emissionen, i.e. Beobachtungen) deuten mit durch das Hidden Markov Modell beschriebenen Wahrscheinlichkeiten auf einen Zustand im System hin.

Die zustandsbasierte Anomalie-Erkennung mittels Hidden Markov Modell lässt sich in folgende Arbeitsschritte gliedern: Beobachtungsphase, Lernphase und Betriebsphase. Im ersten Schritt (der „*Beobachtungsphase*“) werden die Emissionen des Systems zu verschiedenen Zeitpunkten beobachtet. Aus den Emissionen jedes Zeitpunkts werden die für das System markantesten Eigenschaften (Attribute) extrahiert und daraus Samples generiert. Jedes Sample stellt eine Beobachtung zu einem Zeitpunkt dar. Dieser Vorgang findet mehrere Male statt, um das Verhalten des gesamten Systems zu lernen. Am Ende existieren mehrere Datensätze (Sequenzen) mit jeweils einer Abfolge an Samples.

Im zweiten Schritt, der „*Lernphase*“, werden diese gesammelten Daten verwendet um ein HMM zu berechnen. Sie teilt sich in zwei Bereiche auf: Den Initiierungsprozess und den Lernprozess. Bei der Initiierung wird ein HMM-Rohling als Ausgangspunkt für den nachfolgenden Lernprozess gebildet. Das initiale HMM beschreibt die Modellzustände, aber noch keine exakten Wahrscheinlichkeitswerte. Es dient lediglich als Vorlage für den Lernprozess. Der Lernprozess verarbeitet, auf dem initialen HMM basierend, die Datensätze/Samples und berechnet die Anfangs- und Übergangswahrscheinlichkeiten der Zustände. Zum Erlernen der genauen Wahrscheinlichkeitswerte, also der Optimierung des Hidden Markov Modells, wird in der Regel der Baum-Welch-Algorithmus verwendet. Wie Fink [Fink03, S.85ff] beschreibt, optimiert er ein gegebenes (initiales) HMM in Abhängigkeit von den übergebenen Datensätzen. Der Baum-Welch-Algorithmus nähert das initiale HMM über mehrere Iterationen weiter an, bis eine gewünschte Genauigkeit erreicht ist oder sich die Wahrscheinlichkeits-

werte nicht weiter ändern. Die Erzeugung des Hidden Markov Modells ist mit dem vergleichsweise größten Rechenaufwand verbunden.

Wurde ein HMM erstellt, wird in die „Betriebsphase“ (dritter Schritt) übergegangen. In der Betriebsphase wird das HMM zur Evaluierung neuer oder anderer Datensätze verwendet. Eine neue Beobachtungsfolge wird auf gleichem Wege wie in der Beobachtungsphase zu einem Datensatz aus Samples verarbeitet. Diese Datensätze werden dann mit dem HMM verglichen. Das Resultat ist eine Wahrscheinlichkeit, mit welcher der neue Datensatz dem HMM entspricht.

III. Aufbau zur zustandsbasierten Anomalie-Erkennung

Da es sich beim HTTP Protokoll um ein zustandsloses Protokoll handelt, aus dessen Sicht jede HTTP Transaktion eine alleinstehende Aktion ist, können in der Beobachtungsphase keine direkten Zustände ausgelesen werden. Das HTTP Protokoll kennt keine Zustände wie „Angemeldet“, „Administrativer Benutzer“ oder „Session-Timeout“. Diese Informationen werden von der Web-Anwendung erzeugt und verwaltet. Die Zustände der Web-Applikation sind für das HTTP Protokoll und einen Beobachter in erster Linie versteckt. Die Zustände, welche eine Grundvoraussetzung für die Anwendung eines Hidden-Markov-Modells sind, müssen mit Hilfe von sogenannten Emissionen erfasst werden. In den Transaktionen im HTTP Protokoll können Emissionen gemessen werden, welche auf gewisse Vorgänge in der Web-Applikation hindeuten. Aus einer bestimmten Kombination von Emissionen lässt sich der Zustand schlussfolgern. Damit eine Webseite einen Benutzer wiedererkennt wird eine Session-ID erzeugt. Diese ID wird dem Benutzer beim ersten Aufruf zur Verfügung gestellt und von ihm bei jedem weiteren Aufruf erneut zur Identifikation mitgesendet. Anhand dieser Benutzersitzung werden zusammengehörige und aufeinanderfolgende Transaktionen logisch verknüpft und einem Benutzer zugeordnet. Diese Session-ID ist für Beobachter ersichtlich, sofern die Verbindung nicht verschlüsselt ist oder Einsicht in die verschlüsselten Abläufe besteht. Die Session-ID lässt sich für die Zwecke der zustandsbasierten Anomalie-Erkennung nutzen. Eine Benutzersitzung ist gleichbedeutend mit einer Beobachtungssequenz eines Markov Modells. Die Beobachtungssequenz besteht aus Samples, welche jeweils eine Reihe an Emissionswerten beinhalten.

An diesem Punkt stellt sich die Frage, aus welchen Emissionen ein Sample bestehen kann. Zum Ableiten der Emissionswerte eignen sich Parameter des HTTP Protokolls, welche häufig auftreten und auf markante Vorgänge in der Web-Applikation hindeuten. Um die Emissionen einer Beobachtung an eine bestimmte Situation in der Web-Applikation zu binden, werden Pfadinformationen aus der HTTP Transaktion extrahiert und dem Sample hinzugefügt. Dadurch werden wichtige Strukturinformationen der Web-Applikation in die

Beobachtungen integriert. Es empfiehlt sich, die aufgezeichneten Pfade der Requests in logische Gruppen zu teilen, damit sich Emissionen immer an eine Gruppe zusammengehöriger Pfade binden statt an einzelne Pfade. Dies erhöht die Flexibilität des Systems. So werden beispielsweise folgende URLs in eine logische Gruppe zusammengefasst:

- /album/2009/start.htm
- /album/2009/seite1.htm
- /album/2009/seite2.htm
- /album/2009/seite3.htm
- /album/2009/seite4.htm
- /album/2009/seite5.htm
- /album/2010/start.htm
- /album/2010/seite1.htm

Diese URLs haben die gleichen Emissionen und die gleichen Auswirkungen im System. Logisch betrachtet gehören sie alle dem gleichen Zustand an. Der Zustand kann mit „view gallery“ betitelt werden. Ohne das Zusammenfassen dieser Pfade würden für den logisch gleichen Zustand viele verschiedene Samples und Zustände erstellt werden, was zu einer unnötigen Vergrößerung des Markov Modells führt. Im Extremfall gehen relevante Ereignisse in einer Masse an unterschiedlichen Beobachtungen verloren. Für dieses Clustering der Pfade werden im Prototyp Byte-Statistiken aller URLs erstellt und mittels des KMeans-Algorithmus gruppiert.

Zu beobachtende Emissionen lassen sich in zwei Kategorien aufteilen: Jene, welche anwendungsunabhängig implementiert und beobachtet werden können (Allgemeine Emissionen), und jene, welche wertvolle Informationen bieten, aber mitunter anwendungsspezifisch angepasst werden müssen (Spezifische Emissionen). Nachfolgende Emissionen haben sich als aussagekräftige Elemente herausgestellt. Abhängig von der zu beobachtenden Web-Applikation gibt es weitere relevante Emissionen.

Allgemeine Emissionen:

- a. Request-Content-Type: „text/plain“, „multipart/form-data“, „application/x-www-form-urlencoded“, „null“
- b. Verbindungstyp: HTTP, HTTPS, FTP
- c. Request-Methode: GET, POST, PUT, ...
- d. Error-Code der Response: 200 (OK), 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), ...

Spezifische Emissionen:

- e. Informationen über den Anmeldestatus
 - f. Anmelde-Request eines Benutzers
- (a.) Der Request-Content-Typ gibt Aufschluss darüber, ob es sich bei einem Request um einen regulären Request handelt, ob Formularaten übertragen werden oder ob

ganze Dateien in der Payload übermittelt werden (zum Beispiel bei einem File-Upload).

- (b.) Manche Anwendungen stellen Benutzern eine unverschlüsselte Verbindung zur Verfügung und setzen Verschlüsselung nur für bestimmte Bereiche ein. Es ist auch möglich, dass eine Web-Applikation auf FTP-Dateien verweist. Diese Ereignisse lassen sich durch Beobachten des Verbindungstyps auswerten und in Kombination mit anderen Emissionen und der Pfadinformation an eine bestimmte Situation binden.
- (c.) Die Request-Methode bestimmt, in welcher Form Daten an den Server übertragen werden. In einer Web-Anwendung können je nach Zweck des Requests verschiedene Methoden vorkommen.
- (d.) Der Error-Code gibt Aufschluss über fehlerhaftes Benutzerverhalten, sofern es sich um keine Fehlkonfiguration der Web-Applikation handelt. In den meisten Systemen schließen Requests meistens mit dem Response-Code 200 (OK) ab. Je nach System kommt es an manchen Stellen im System zu einem Response-Code 302 (Found). Dies tritt auf, wenn durch das „Location“-Header-Feld ein Redirect auf eine andere Seite ausgelöst wird. Der Response-Code 403 (Forbidden) gibt, bei korrekter Anwendungskonfiguration, Auskunft über unerlaubtes Benutzerverhalten, zum Beispiel wenn ein Benutzer durch manuelle URL-Eingabe versucht, auf Dateien zuzugreifen.
- (e.) Informationen über den Anmeldestatus ermöglichen das Lernen der Verhaltensmuster von angemeldeten und abgemeldeten Benutzern. Angemeldete Benutzer dürfen in vielen Web-Anwendungen auf mehr Dateien zugreifen und mehr Aktionen ausführen als nicht angemeldete Benutzer. Der Anmeldestatus lässt sich nicht durch ein Attribut einer HTTP Transaktion auslesen, daher ist eine Funktion nötig, welche diesen Status zum Beispiel aus der Response auswertet (Logout-Link vorhanden? Bestimmtes Cookie existent? Sonstige Flags?)
- (f.) Das Feststellen eines Anmeldeversuchs bietet ebenfalls interessante Informationen. Eine Web-Anwendung kann so konfiguriert sein, dass auf einen erfolgreichen Anmeldeversuch eine Willkommenseite erscheint. Bei einem fehlerhaften Anmeldeversuch erscheint die Login-Seite erneut. Die Wahrscheinlichkeit, dass auf eine Login-Seite die Willkommenseite folgt, ist in diesem Fall sehr hoch. Der Wechsel von Login-Seite auf Login-Seite ist hingegen mit geringer Wahrscheinlichkeit gegeben. Wenn ein Angreifer versucht, das Passwort eines Benutzers zu erraten und mehrere Anmeldeversuche tätigt, tendiert die Wahrscheinlichkeit für sein Verhalten durch die Fehlversuche rasch gegen Null und der Angreifer wird erkannt. Auch diese Emission benötigt eine spezifische Funktion, welche auswertet ob es sich bei einem Zustand um eine Login-Seite handelt.

Abbildung 1 zeigt eine Sammlung an Sequenzen von

Samples. Jede Zeile stellt eine Benutzersitzung dar, wobei jeweils zwischen eckigen Klammern der Sample-Vektor dargestellt wird. Der Wert vor dem Strichpunkt entspricht dem Pfad-Cluster.

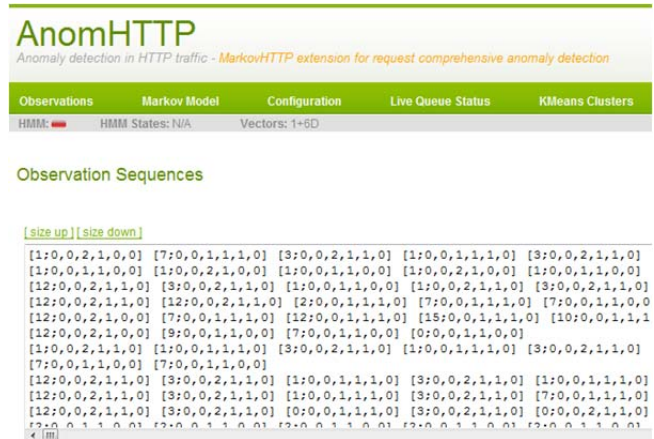


Abbildung 1 – Beobachtete Sequenzen und Samples

IV. Implementierung eines Prototyps

Der in dieser Publikation vorgestellte Ansatz zur zustandsbasierten Anomalie-Erkennung nutzt das Hidden Markov Modell zur Auswertung von Anomalien im Benutzerverhalten und wurde als Prototyp implementiert. Bei dem Prototyp handelt es sich um einen in Java implementierten Reverse-Proxy, der Requests für eine Webseite entgegennimmt, auswertet und in einer gesonderten Verbindung in Vertretung für den Antragsteller an den Web-Server weiterleitet. Die Response des Web-Servers wird in umgekehrter Richtung über den Proxy an den Antragsteller zurückgesendet. Diese Reverse-Proxy-Funktionalität wird bereits durch den Prototyp „AnomHTTP“ von Kirchner [Kirc10] zur Verfügung gestellt. AnomHTTP unterstützt HTTP sowie HTTPS Verbindungen. Für die Verarbeitung von HTTPS-verschlüsselten Verbindungen muss AnomHTTP als Reverse-Proxy diese Verbindungen an Stelle des Web-Servers terminieren. Es bietet sich an, AnomHTTP als Grundlage für die Implementierung des in dieser Arbeit vorgestellten Prototyps zu erweitern. Als Bibliothek, welche die Funktionen im Zusammenhang mit Hidden Markov Modellen zur Verfügung stellt, kommt „Jahmm“ zum Einsatz. Jahmm umfasst unter anderem eine Implementierung der Viterbi-, Forward-Backward-, Baum-Welch- und KMeans-Algorithmen [JeFr06][Fink03].

In der Lernphase der zustandsbasierten Anomalie-Erkennung muss, ähnlich der herkömmlichen Anomalie-Erkennung, das Normalverhalten der Web-Applikation gelernt werden. Es wird gelernt welche Zustände es gibt, welche Zustände am Häufigsten als Anfangspunkte für eine Benutzersitzung dienen und mit welchen Wahrscheinlichkeiten zwischen den gelernten Zuständen gewechselt werden kann. Dafür werden die Benutzersitzungen auf Anwendungsebene anhand der Session-ID ausgewertet, wichtige Attribute extrahiert und

Samples erzeugt. Mit Hilfe dieser Samples wird ein Hidden Markov Modell berechnet, welches das Normalverhalten der Web-Anwendung beschreibt. Zukünftige Benutzersitzungen können durch dieses Modell parallel zur Request-Übertragung auf Legitimität überprüft werden. Basierend auf festgelegten Schwellwerten kann ein untypischer Zustandswechsel sofort vom Prototyp erkannt und blockiert oder zur weiteren Auswertung gespeichert werden. Im Prototyp werden identische Samples (Vektor aus Attributen) in einen repräsentativen ganzzahligen Wert umgewandelt. Im HMM wird mit diesen ganzzahligen Werten gerechnet. Eine Sample-Ausprägung (Kombination aus Attributen) steht für einen bestimmten Zustand im System (gültig sowie ungültig). Bei der initialen Berechnung des Hidden Markov Modells entstehen so viele Zustände wie es unterschiedliche Samples gibt. Durch den Baum-Welch-Algorithmus werden die Übergangswahrscheinlichkeiten zwischen den Zuständen sowie deren Anfangswahrscheinlichkeiten berechnet. Das Resultat ist ein HMM, welches das Benutzerverhalten in der Web-Applikation spiegelt. Bei der Beobachtung des Benutzerverhaltens ist es nicht wichtig zu erkennen, ob ein Benutzer in einen untypischen Zustand wechselt. Ob ein Zustand typisch ist, wird aufgrund der im HMM berechneten Übergangswahrscheinlichkeit zum Ausdruck gebracht. Untypische Zustände treten während der Beobachtung im Vergleich zu typischen Zuständen selten auf. Wird ein Zustand während der Beobachtungsphase selten aufgezeichnet, ergibt sich im HMM eine geringe Wahrscheinlichkeit für einen Übergang in diesen Zustand. Untypische Zustände entsprechen mit hoher Wahrscheinlichkeit einer Anomalie. In manchen Fällen kann es sich allerdings um einen gültigen Zustand handeln, welcher lediglich selten auftritt. In diesen Fällen ist es nötig, in das Hidden Markov Modell einzugreifen und Spezialfälle als gültig zu markieren. Da es sich bei der zustandsbasierten Anomalie-Erkennung um ein positives Sicherheitsmodell handelt, liegen die genannten Spezialfälle in endlichem Umfang vor. Abhängig von dem Umfang der Web-Applikation ist davon auszugehen, dass die Häufigkeit untypischer aber dennoch normaler Zustandsübergänge durch manuelle Eingriffe exponentiell sinkt.

In der Betriebsphase wird das Verhalten der Benutzer weiterhin beobachtet. Aus den neuen Beobachtungen werden wie in der Lernphase Samples generiert. Diese neuen Samples werden einerseits in der Datenbank dauerhaft gespeichert um für Neuberechnungen/ Aktualisierungen des Hidden Markov Modells zur Verfügung zu stehen. Andererseits werden die letzten x Samples¹ jeder aktiven Benutzersitzung in der „Live-Queue“ gespeichert. Die zustandsbasierte Anomalie-Erkennung erfolgt anhand dieser x Beobachtungen. Durch die Beschränkung auf die letzten Samples einer Benutzersitzung entstehen Auswertungsfenster gleicher Größe, die einen Vergleich untereinander ermöglichen.

¹ Die Anzahl der zu haltenden Elemente wird in der Konfiguration des Prototyps eingestellt.

Andernfalls tritt das Problem auf, dass manche Benutzersitzungen sehr kurz und andere beliebig lange werden und die daraus berechneten Wahrscheinlichkeiten nicht miteinander vergleichbar sind. Mit der Zunahme der Sequenzlänge sinkt die Wahrscheinlichkeit der Sequenz gegen 0. Die Auswertungsfenster können auch mit denen anderer Benutzersitzungen verglichen werden, um beispielsweise einen üblichen Wert für die Wahrscheinlichkeit eines solchen Fensters im vorliegenden System zu definieren. Durch die Einführung von Auswertungsfenstern ergeben sich zudem weitere Möglichkeiten für die zustandsbasierte Anomalie-Erkennung. Es kann zum Beispiel ein Grenzwert definiert werden, unterhalb dessen das Benutzerverhalten als Anomalie gewertet wird. Eine Denial of Service Attacke oder ein Störungsversuch kann festgestellt werden, falls ein Auswertungsfenster mit gleichen Samples gefüllt ist.

Für jede Benutzersitzung in der Live-Queue gibt es eine Gewichtungvariable. Wird in einem Auswertungsfenster einer Benutzersitzung ein anomales Verhalten festgestellt, wird dieses in der Gewichtungvariable festgehalten. So wird sichergestellt, dass erkannte Anomalien nicht verloren gehen, wenn sie aus dem Beobachtungsfenster verdrängt werden. Um die Wahrscheinlichkeit des Benutzerverhaltens zu erhalten, muss die Wahrscheinlichkeit des Auswertungsfensters mit dem Wert der Gewichtungsvariable multipliziert werden. Wenn viele Anomalien erkannt werden, tendiert die Gewichtungsvariable gegen 0. Die Wahrscheinlichkeit des Benutzerverhaltens wird direkt davon beeinflusst und sinkt in Relation dazu. Abbildung 2 stellt den Aufbau der Live-Queue grafisch dar.

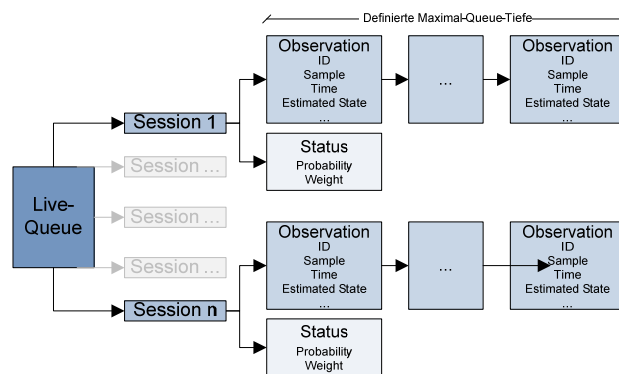


Abbildung 2 – Aufbau Live-Queue

V. Testergebnisse des Prototyps

Zum Testen des Prototyps wird das Protokoll der Verwendung einer Web-Applikation benötigt. Die Aufzeichnung muss die vollständigen Inhalte der HTTP Transaktionen enthalten, damit alle nötigen Emissionen extrahiert werden können. Datensätze, die aus einem Produktivsystem stammen, sind gegenüber manuell generierten vorzuziehen. Dadurch wird sichergestellt, dass die Testergebnisse mit einer realen Umgebung vergleichbar sind. Zusätzlich ist die Veröffentlichung dieser Datensätze sinnvoll, damit Dritte die Ergebnisse eigener Tests

vergleichen können. Durch diese Faktoren ist es schwierig an geeignete Datensätze zu kommen. Datensätze aus einem Produktivsystem enthalten vertrauliche Inhalte und dürfen aus Datenschutzgründen nicht öffentlich zugänglich gemacht werden. In den Requests und Responses vollständiger Datensätze sind Benutzernamen, Passwörter und unter Umständen weitere persönliche Daten enthalten. Da keine passenden Datensätze für die Evaluierung des Prototyps zugänglich sind, müssen eigene Daten gesammelt werden. In Zusammenarbeit mit dem Administrator einer Community werden im Produktivbetrieb der Community Daten für die Evaluierung gesammelt. Bei der Community handelt es sich um eine Web-Applikation mit insgesamt 35994 Mitgliedern (seit 2002) und durchschnittlich 10 gleichzeitig aktiven Besuchern. In der Web-Applikation werden Fotos und Informationen zu Veranstaltungen für Benutzer bereitgestellt. Jeder Benutzer verfügt über eine eigene Benutzerseite mit Profil und Kommunikationsmöglichkeiten wie Gästebuch und privaten Nachrichten.

Durch die Protokollierung der Web-Applikation stehen Datensätze im Umfang von 424.040 HTTP Transaktionen zur Verfügung. Die Daten verteilen sich auf 871 Benutzersitzungen. Aus diesen vollständigen HTTP Transaktionen des Beobachtungszeitraums müssen irreführende Einträge wie zum Beispiel automatisierte Requests oder Daten, welche üblicher Weise im Browser-Cache gespeichert bleiben entfernt werden. Das Verhalten, wie lange welche Dateien im Cache gehalten werden, variiert von Browser zu Browser und ist teilweise auch von der Konfiguration abhängig. Durch unvorhersehbares Caching-Verhalten kommt es aus Sicht des Hidden Markov Modells zu zufälligen Übergängen, da manchmal auf den gleichen Request einer HTML-Datei ein Bild-, Stylesheet- oder Javascript-Request folgt und manchmal nicht. Des Weiteren ist ein Hidden Markov Modell, wie bereits erwähnt, jeweils nur vom vorigen Zustand abhängig. Wenn in einer Web-Applikation nach jedem Request automatisch ein Request für das zugehörige Stylesheet oder die Javascript-Datei erfolgt, wird die Funktionalität des Hidden Markov Modells beeinträchtigt. Daher ist es wichtig, möglichst nur HTTP Transaktionen auszuwerten, welche das Benutzerverhalten wiedergeben. Aussagekräftige Benutzeranfragen lassen sich am besten durch den Content-Type der zugehörigen Response (nicht zu verwechseln mit dem Request-Content-Type) filtern („text/html“, „text/plain“, „text/xml“,...). Nach dem Entfernen verbleiben 36.348 Transaktionen/Samples zum Trainieren eines Hidden Markov Modells. Die starke Reduzierung kommt dadurch zustande, dass in der Web-Applikation viele HTTP Transaktionen durch Werbung und Statusabfragen automatisiert ausgelöst werden.

Für die Evaluierung des Prototyps werden die Pfadinformationen in 20 Gruppen sortiert. Bei der Erstellung des Hidden Markov Modells ergeben sich dadurch und durch die aufgezeichneten Samples 52 Modellzustände. Die konfigurierte Größe der Live-Queue beträgt 10 Einträge pro Benutzersitzung. Wird ein unbekannter Zustand beobachtet, wird die Gewichtung

variable mit einem Wert von 0.00000002 multipliziert. Ein Beobachtungsfenster voller identischer Elemente fließt mit einem Faktor von 0.0000003 in die Gewichtungvariable ein. Ein Zustandswechsel von einem Zustand auf sich selbst mit einer Wahrscheinlichkeit unter 0.005 wird mit seiner eigenen Auftrittswahrscheinlichkeit festgehalten. Die Gesamtwahrscheinlichkeit des Beobachtungsfensters fließt, wenn sie den Schwellwert von 0.0000000002 unterschreitet, mit einem Gewichtungswert von 0.0000002 ein. Die genannten Werte können in der Konfiguration des Prototyps angepasst werden. Zur Evaluierung des Prototyps wird Benutzerverhalten simuliert, welches durch die Live-Queue des Prototyps auf Anomalien überprüft wird. Die Live-Queue schreibt Log-Daten und hält darin typisches Verhalten und Abweichungen fest.

Abbildung 3 stellt die von der Live-Queue nach jeder Benutzerinteraktion berechneten Wahrscheinlichkeiten des Beobachtungsfensters (ab Schritt 10) graphisch dar. Ab der zehnten Beobachtung in einer Benutzersitzung ist die Beobachtungsfenstergröße ausgefüllt (mit jeder weiteren Beobachtung wird die älteste Beobachtung aus dem Beobachtungsfenster verdrängt) und die Wahrscheinlichkeiten sind vergleichbar zu nachfolgenden Werten. Dabei zeigt sich, wie Wahrscheinlichkeiten bei typischem Benutzerverhalten um einen bestimmten Mittelwert herum auf und ab schwanken, je nachdem wie wahrscheinlich die letzten zehn Benutzeraktionen sind.

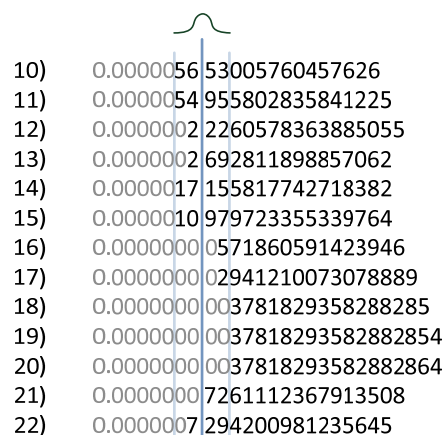


Abbildung 3 – Wahrscheinlichkeiten Auswertungsfenster

Nachfolgender Log-Auszug (Tabelle 1) zeigt das Absinken der Wahrscheinlichkeit nach dem Auftritt einer untypischen Benutzeraktion in Schritt 10. Das untypische Verhalten wird in der Gewichtungvariable („Weight“) dauerhaft mit dem Faktor 0.0000002 festgehalten und zukünftig (ab Schritt 11 in diesem Beispiel) in die Auswertung einbezogen.

I. Ausblick

Für die Weiterentwicklung des Prototyps ist die Integration einer Möglichkeit zur administrativen Benutzerinteraktion, um untypisches aber legitimes Verhalten („false positives“) zu bewerten, ein nötiger Arbeitsbereich. Ein weiterer Ansatzpunkt für eine Weiterentwicklung ist das für

Literatur

- [DeDo86] D. E. Denning: An Intrusion Detection Model. In: Proceedings of the Seventh IEEE Symposium on Security and Privacy (1986). Seite 130.
- [Fink03] G. A. Fink: Mustererkennung mit Markov-Modellen – Theorie – Praxis – Anwendungs-gebiete. Teubner (2003).
- [Kirc10] M. Kirchner: Detection of Attacks and Anomalies in HTTP Traffic Using Instance-Based Learning and KNN Classification. Diplomarbeit, FH Hagenberg (2010).
- [Mods11] ModSecurity. <http://www.modsecurity.org>. Aufgerufen: 16.04.2011.
- [Sieg05] C. Siegel: Markov-Ketten. In: Facharbeit Mathematik, Version 2005. <http://www.siegel-christian.de/media/facharbeit/Markov-ketten.pdf>. Aufgerufen: 16.04.2011.
- [Snor11] Snort. <http://www.snort.org>. Aufgerufen: 16.04.2011.
- [Unip11] Uni-Protokolle.de: www.uni-protokolle.de – Die Adresse für Ausbildung, Studium und Beruf. http://www.uni-protokolle.de/Lexikon/Stochastischer_Prozess.html. Aufgerufen: 16.04.2011.
- [JeFr06] F. Jean-Marc: Jahmm – An implementation of HMM in Java. Markkov Bibliothek für Java. Beispiele, Dokumentation und Source Code. <http://code.google.com/p/jahmm>. Aufgerufen: 20.05.2011.
- [ArGP07] D. Ariu, G. Giacinto, R. Perdisci: Sensing attacks in Computers Networks with Hidden Markov Models. In: MLDM '07 Proceedings of the 5th international conference on Machine Learning and Data Mining in Pattern Recognition (2007).
- [Vamo11] B. Vamos: Zustandsbasierte Anomalie-Erkennung im Benutzerverhalten basierend auf dem HTTP Protokoll unter Verwendung von Markov Ketten. Diplomarbeit, FH Hagenberg (2011).