

Anomalieerkennung in Webanwendungen

Sebastian Schlein

Matthias Balke

Technische Universität Dortmund
Proseminar Sicherheit in Webanwendungen

Betreuer: Christian Bockermann

10. August 2008

Inhaltsverzeichnis

1	Einleitung	3
2	Erkennungsmöglichkeiten	4
2.1	Intrusion-Detection-Systeme	4
2.2	Web-Application-Firewalls	5
2.3	Anomalie-Erkennungs-Systeme	5
3	Vorbereitung	5
4	Strategien	6
4.1	Attributlänge	7
4.2	Buchstabenverteilung	7
4.3	Strukturelle Anomalien	8
4.4	Token Finder	9
4.5	Attributpräsenz	10
4.6	Reihenfolge	10
4.7	Zugriffshäufigkeit	11
4.8	Anfrageverzögerung	12
4.9	Interaktionsreihenfolge	13
5	Analyse	13
6	Fazit	14

1 Einleitung

Heutzutage befinden sich wichtige und sensible Daten auf Computern, die mit dem Internet verbunden sind. An diese Daten zu kommen ist das Ziel von Angreifern, die versuchen sich mit Hilfe dieses Wissens einen eigenen Vorteil zu verschaffen, indem sie die Daten für sich nutzen.

Viele Informationen sind in Datenbanken auf Servern gespeichert und durch Webanwendungen über das Internet abrufbar. Dadurch ist es möglich, dass Unternehmen von überall auf ihre Daten zugreifen, sie aber zentral an einem Ort sichern und verwalten können. Diese Technik ermöglicht das Erstellen von Anwendungen, die zentral laufen und von überall aus erreichbar sind. Sie müssen in der Regel nicht extra installiert oder eingerichtet werden und funktionieren somit ohne Aufwand von jedem Arbeitsplatz auf der Welt, sobald ein Internetzugang zur Verfügung steht.

Im Zuge der Entwicklung von Anwendungen wie Foren, Onlineshops, Gästebüchern, Mailsystemen und Sozialen Netzwerken treten immer mehr Sicherheitsprobleme auf. Ursprünglich bestand die größte Gefahr aus Computerviren, die über Datenträger weitergegeben wurden. Durch das Internet verbreiteten sich allerdings nicht nur Viren, deren Ziele es eher waren Informationen zu vernichten, sondern auch solche, die nur darauf aus sind, Informationen zu sammeln und an den Angreifer zu übermitteln.

Seit es Webanwendungen gibt, existieren also neue Angriffsmethoden, die genutzt werden können, um direkt an Daten zu kommen, ohne Viren oder Ähnliches einzuschleusen. Heutzutage finden 20-30 Prozent der Angriffe über das Netz statt und daher werden Möglichkeiten benötigt, mit denen diese Angriffe schnell und zuverlässig zu erkennen sind, damit Gegenmaßnahmen ergriffen werden können.

Die bisherigen Möglichkeiten der Angriffserkennung setzen auf Web-Application-Firewalls, Abstract-Security-Modelle und Intrusion Detection Systeme (IDS). Diese erkennen Angriffe über spezielle Angriffsmuster und -signaturen oder vergleichen Anfragen mit vorher aufwändig erstellten Regelwerken, die zuvor in die Konfiguration mit einfließen. Diese Sicherheitsmaßnahmen haben allerdings den Nachteil, dass sie für selbstentwickelte Anwendungen schwer zu warten und konfigurieren sind. Da diese Wartung von Spezialisten gemacht werden muss und zudem noch zeitnah nach Fertigstellung der Applikation erfolgen sollte, sind diese Erkennungsmethoden verhältnismäßig teuer und zeitaufwendig. Durch die ständige Entwicklung neuer Angriffe, erhöht sich der Aufwand enorm, da die Sicherheitstechniken ebenfalls immer aktuell sein sollten.

Gebraucht wird also ein Sicherheitssystem, das sich automatisch an jede beliebige Anwendung anpasst und möglichst wenig Konfigurationsaufwand benötigt. Hier setzt das Anomalieerkennungsverfahren an, welches das Verhalten einer Webanwendung automatisch analysiert und dann erkennen kann, wenn ein Angriff stattfindet. In der folgenden Arbeit wird der Aufbau und die Arbeitsweise eines solchen Systems anhand einer Stu-

die [3] von Christopher Kruegel und Giovanni Vigna von der University of California beschrieben.

2 Erkennungsmöglichkeiten

Um einen Angriff auf eine Webanwendung zu erkennen, müssen Attacken von normalem Nutzerverhalten unterschieden werden können. Dazu gibt es mehrere Techniken, die durch statistische Verfahren und Analysen des Netzwerkverkehrs erkennen können, ob die Anwendung grade normal genutzt oder angegriffen wird.

Angriffe lassen sich zum Beispiel durch stark erhöhten Netzwerkverkehr oder eine deutlich höhere Auslastung des Systems erkennen. Steigt die Anzahl der Datenbankanfragen bei gleicher Anzahl der Benutzer stark an, ist dies ein Zeichen dafür, dass vermehrt Anfragen an das System gestellt werden, die in der Masse so nicht vorgesehen sind.

Ein Beispiel hierfür wäre ein Angriff durch ein Botsystem, welches sehr schnell viele unterschiedliche Seiten aufruft, um an die Informationen zu kommen, die dort hinterlegt sind. Ein Data-Mining-Bot könnte beispielsweise in einem Social Network Benutzerprofile anfragen und über die auf dem Benutzerprofil hinterlegten Links zu andere Profilen automatisch neue Anfragen über die nun neu erkannten Benutzer stellen. Da dieser Bot die Informationen auf den Seiten deutlich schneller als ein Mensch verarbeiten und zusätzlich noch mehrere Profile gleichzeitig betrachten kann, lässt sich durch einfache statistische Analysen erkennen, dass dies kein gewünschtes Verhalten ist und es sich daher um eine Anomalie handelt.

Um nun ein System aufzubauen, welches erkennt, dass es angegriffen wird, benutzt man häufig Intrusion-Detection-Systeme, die anhand von vorher festgelegten Regeln das Verhalten in der Webanwendung analysieren und beurteilen.

2.1 Intrusion-Detection-Systeme

Intrusion-Detection-Systeme, kurz IDS, dienen zur Erkennung von Angriffen auf eine Webanwendung. Für diese Erkennung gibt es mehrere Ansätze, die auf unterschiedliche Art und Weise funktionieren und bei einem Angriff Maßnahmen ergreifen, um den Angriff abzuwehren.

Zuerst müssen IDS und Anwendung aneinander angepasst werden, dann folgt die Konfigurationsphase, in der Angriffe durchgespielt und auf dessen Grundlagen Signaturen entworfen werden, wie ein möglicher Angriff aussehen könnte. Diese Regeln und Signaturen sind fest im IDS verankert und brauchen umständliche Pflege. Sie müssen immer auf dem neuesten Stand sein, damit einen möglichst hoher Schutz gewährleistet werden kann. Es lässt sich zwar erkennen, dass Intrusion Detection eine wirkungsvolle Möglichkeit ist, Angriffe zu erkennen und zu verhindern, jedoch ist sie auch sehr zeit- und kostenaufwendig, da die Wartung von Spezialisten durchgeführt werden muss.

2.2 Web-Application-Firewalls

Ein weiteres Schutzsystem ist die Web-Application-Firewall (WAF). Sie ist als Reverse-Proxy implementierbar und schützt somit nicht nur eine Webanwendung, sondern kann mehrere verteilte Anwendungen überwachen. Dadurch verringert sich der Konfigurationsaufwand des Sicherheitssystems und es kann unabhängig von den zu schützenden Anwendungen gewartet und installiert werden.

Als weiteren Vorteil kann man die Tatsache betrachten, dass durch die lose Kopplung von Firewall und Webanwendung auch Anwendungen von Drittanbietern mit geschützt werden, bei denen man sich nicht sicher sein kann, ob sie den Sicherheitsstandards der eigenen Systeme entsprechen.

Bei der Ein- und Ausgabe von Daten durch die Benutzer oder die Webanwendung, filtert die Firewall diese Datenströme und überprüft anhand von vorher festgelegten Regeln, ob in den Daten unerwünschte Zeichenketten vorkommen, die einen Angriff auslösen könnten.

Leider ist dieses Filtersystem verhältnismäßig aufwendig, erfordert daher viel Rechenleistung und blockiert bei einem Ausfall die komplette Kommunikation zu den zu schützenden Anwendungen.

2.3 Anomalie-Erkennungs-Systeme

Anomalie-Erkennungs-Systeme (ADS) überprüfen mittels mathematischer Verfahren die Parameter, die bei Benutzereingaben in Webanwendungen entstehen. Weichen diese Parameter weit genug von der Norm ab, wird ein Angriff erkannt und kann verhindert werden.

Im Verlauf dieser Arbeit wird ein System betrachtet, welches automatisch lernt und deshalb nicht gewartet werden muss. Der Konfigurationsaufwand, der bei IDS und Web-Application-Firewalls entsteht, entfällt und vereinfacht somit die Installation erheblich.

3 Vorbereitung

Um Angriffe mit einem Anomalie-Detection-System erkennen zu können greifen Kruegel und Vigna auf die automatisch vom Server generierten Logs zurück. Dies erfordert kein Eingreifen in die eigentliche Webanwendung und die Daten stehen immer zur Verfügung.

Der Vorteil der Serverlogs ist, dass es zwar unzählige Webanwendungen, aber nur eine Hand voll benutzter Webserver gibt, die alle standardisierte Logfiles generieren. Als Eingabe für das ADS werden also Logeinträge des Servers benutzt. Diese Logeinträge müssen so gefiltert werden, dass nur noch die über bleiben, die von dem System ausgewertet werden können und für eine Anomalie überhaupt in Frage kommen. Zuerst beschränken sie den Filter auf Anfragen, die einen HTTP-Status zwischen 200 und 300 aufweisen und damit erfolgreich verlaufen sind. Weiter werden nur GET Anweisungen

betrachtet, da diese Parameter beinhalten und nur diese für eine mögliche Anomalie in Frage kommen. Jetzt steht dem ADS eine Liste zur Verfügung, die ausgewertet und nach Anomalien durchsucht werden kann.

Beispiel einer Zeile aus der Liste:

```
127.0.0.1 - johndoe [23/Jun/2008:14:30:54 -0100] "GET /application/login.php?user=admin&password=abc" 200
```

Dieser String beinhaltet die IP-Adresse, den Zeitpunkt, das aufgerufene Skript und die Parameter, die der Anfragende übermittelt. Er könnte von einem Login generiert worden sein, bei dem sich der Benutzer "admin" mit dem Passwort "abc" in eine Webanwendung einloggen möchte. Interessant für die Anomalieerkennung ist in erster Linie die Parameterübergabe, der so genannte Querystring, der ab dem "?" beginnt und die Werte beinhaltet, die der potenzielle Angreifer übermitteln will. Die Parameter bestehen aus Tupeln (x, y) mit x als Variablennamen und y als Wert. Somit wäre "user" eine Variable, die den Wert "admin" enthält. Weitere Werte, die in die Analyse mit einfließen sind die IP und der Zeitpunkt. Sie dienen dazu Anfragen einem bestimmten Computer zuzuweisen und eindeutig zu erkennen. Die IP-Adresse beschreibt dabei eine Adresszuweisung eine Internetproviders an einen Kunden, der durch den ebenfalls im Eintrag enthaltenen Zeitpunkt eindeutig identifiziert werden kann.

Am Beispiel dieses Logeintrags sieht man, wie gebündelt die Informationen zur Verfügung stehen, die dann nur noch ausgewertet werden müssen. Es ist also nicht nötig eine besondere Schnittstelle für eine Anwendung zu entwickeln, die dann mit dem ADS zusammenarbeitet. Dadurch ist es auch möglich das System nachträglich zu installieren, ohne dass die Webanwendung angepasst werden muss.

Einige Auswertungsmethoden, mit denen Angriffe erkannt werden können, werden im folgenden Abschnitt behandelt.

4 Strategien

Bei einem Angriff auf eine Webanwendung manipuliert der Angreifer die Parameter, die einer Anfrage an die Anwendung übergeben werden. Daher benötigt das Anomalie-Detection-System Modelle, an denen es entscheiden kann, ob ein Request gültig ist, oder einen Angriff darstellt. Da diese Modelle erst erstellt werden müssen, besitzt das ADS zwei unterschiedliche Modi, in denen es operieren kann.

Im Learning-Mode erstellt es Modelle, die während der Arbeitsphase, dem Detection-Mode, mit Anfragen an den Server verglichen werden. Mit diesen Modellen kann berechnet werden, ob eine Anfrage normal oder feindlich ist. Dazu errechnet das System einen Anomaly-Score, über den es möglich ist einzuschätzen, ob es sich um einen Angriff handelt oder nicht. Die Berechnung des Anomaly-Scores wird mit Hilfe der gewichteten Summe

$$AnomalyScore = \sum_{m=0}^n w_m * (1 - p_m)$$

durchgeführt, wobei m die Anzahl der Modelle, w_m für das Gewicht des Modells und p_m für die Wahrscheinlichkeit besteht, dass es sich um eine Anomalie handelt. Befindet sich der Wert außerhalb der Toleranzgrenzen, handelt es sich um eine Anomalie. Durch die Auswertung über eine Summe von verschiedenen Modellen ist es möglich durch mehrere erkannte Anomalien einen sehr hohen Anomaly-Score zu erzeugen. Bei nur geringer Abweichung an wenigen, nicht eindeutigen, Parametern entsteht so keine Falschmeldung. Im nun folgenden Teil findet sich die Beschreibung der wichtigsten Erkennungsansätze und deren Funktionsweisen.

4.1 Attributlänge

Die Länge eines Attributes ist eines der einfachsten und wirkungsvollsten Überprüfungs-kriterien, da viele Bezeichner eine feste oder zumindest maximale Anzahl an Zeichen haben. Beispiele für Parameter mit einer festen Zeichenanzahl sind Session- und Benutzerids, Datumsanzeigen oder Kreditkartennummern, da diese automatisch generiert werden, oder standardisiert sind. Weicht ein Parameter nun von der vorgegebenen Länge ab, erkennt das System, dass er manipuliert sein muss.

Schwieriger ist diese Erkennung bei Freitextfelder, wie Kommentaren oder Nachrichtensystemen, da die Anzahl der Zeichen dort stark schwanken kann. Daher werden während des Lernprozesses Obergrenzen für diese Parameter ermittelt, die bei signifikater Überschreitung ebenfalls einen Alarm auslösen.

Lernphase Während der Lernphase werden für alle Parameter obere Schranken für die Länge errechnet, die dann in der Erkennungsphase überprüft werden können.

Erkennung Wenn das System Benutzereingaben überprüft, wird mittels der Chebychev Ungleichung [2] bestimmt, ob ein Wert gültig ist oder es sich um einen Angriff handelt. Die Ungleichung berechnet schwache Schranken, die nur bei großen Abweichungen einen Alarm auslösen. Da die Schranken bei variablen Parameterlängen nicht sehr stark sind, resultieren durch diese Überprüfungsmethode nur selten fälschlicherweise ausgelöste Alarme.

4.2 Buchstabenverteilung

Die Buchstabenverteilung basiert auf der relativen Häufigkeit, mit der Buchstaben innerhalb eines Parameters auftreten. Dieses Modell macht sich zu nutze, dass die meisten Werte von Menschen lesbar sind und daher in der Regel nur aus Buchstaben, Zahlen und einigen Sonderzeichen bestehen. Versucht nun ein Angreifer mittels eines Buffer Overflows Teile des Quelltextes zu überschreiben, muss er dies mit Binärdaten tun. Diese Binärdaten weisen jedoch eine komplett andere Struktur als normaler Text auf und werden

dadurch als Anomalie erkannt. Ein ähnlicher Angriff, der auch direkt durch die veränderte Struktur erkennbar wäre, ist Path-Traversal, da dort einzelne Zeichen ebenfalls mit sehr großen Wahrscheinlichkeiten auftreten.

Berechnet wird die Verteilung mit Hilfe relativer Wahrscheinlichkeiten für einzelne Buchstaben, die dann sortiert und vom eigentlichen Zeichen losgelöst betrachtet werden. Analysiert das Modell die Zeichenkette "aabcde" über der Standard ASCII Kodierung, die 256 Zeichen beinhaltet, ergibt sich daraus die Häufigungsfolge "0,33 0,17 0,17 0,17 0,17 0". In der Zeichenfolge tritt die 0 251 mal auf, da die übrigen 251 Zeichen nicht benutzt wurden. Das normale Verhalten der Häufigungsfolge sieht so aus, dass die Wahrscheinlichkeiten bei längeren Zeichenketten und Texten langsam sinken müssten, da es keine Buchstaben gibt, die extrem oft benutzt werden.

Anomalien treten bei diesem Modell in zwei Fällen auf. Entweder, wenn die Wahrscheinlichkeiten extrem schnell absinken und dadurch einige Zeichen sehr oft vorkommen müssen (Bufferoverflow, Path Traversal), oder wenn die Zeichenverteilung sich fast gar nicht ändert, da die Eingaben zufällig sind. Wird ein Path-Traversal Angriff verübt, tritt der Punkt sehr häufig auf, da versucht wird in ein übergeordnetes Verzeichnis zu wechseln. Diese hohe Konzentration auf einem Wert wird dann als Anomalie erkannt.

Lernphase In der Lernphase wird zuerst eine ideale Verteilung bestimmt. Dazu speichert das System die Verteilungen jeder Anfrage und summiert diese letztendlich so auf, dass sich ein Modell bildet. Dieses Modell der idealen Verteilung ist wohldefiniert, da alle individuellen Variationen, die vorher berechnet wurden, mit eingeflossen sind.

Erkennung In der Erkennungsphase muss geprüft werden, ob die Zeichenverteilung des Parameters von der vorher berechneten idealen Verteilung abweicht. Dafür wird der Pearson X^2 Test [1] heran gezogen. Das gesamte Intervall wird in kleinere Abschnitte verteilt und danach kann entschieden werden, ob der untersuchte Parameter von dem in der Lernphase berechneten Modell abweicht oder nicht.

4.3 Strukturelle Anomalien

Auch wenn viele Angriffe mit den beiden vorher erklärten Methoden identifiziert werden können, ist es möglich Schadcode in Parametern so zu verstecken, dass es den Anschein hat, es handle sich um ganz normale Eingaben. Dazu benutzt man zum Beispiel eine andere Kodierung um Sonderzeichen als Text zu verschlüsseln und kann dadurch unerwünschte Ergebnisse erzielen. Cross-Site-Scripting Angriffe basieren oft darauf, dass zwar Klammern von der Anwendung entfernt, diese aber nur erkannt werden, wenn sie in einer passenden Kodierung vorliegen. Wenn der Angreifer eine andere benutzt, können sie so in die Anwendung gelangen.

Diese Anomalien lassen sich finden, indem für jeden Parameter eine Struktur entworfen wird, die während der Erkennungsphase alle gültigen Eingaben als solche erkennt und

Ungültige klassifizieren kann. Das Problem ist, dass es keine eindeutige Attributbeschreibung gibt, die alle Eingaben eindeutig zuordnen kann und daher nur eine Annäherung an eine Struktur möglich ist, die eine wahrscheinliche Einordnung möglich macht.

Lernphase In der Lernphase wird eine reguläre Grammatik berechnet, die alle erlaubten Vorkommen von Werten in Parametern erzeugen kann. Allerdings werden ungültige Werte benötigt, damit die Grammatik nicht nur die trainierten Daten, sondern auch in der Lernphase nicht vorkommende Werte als normal einstuft. Um diesen schmalen Grad zwischen beiden Extremen zu finden, verwendet man Markov Modelle und die bayessche Wahrscheinlichkeit. Als erstes wird eine Wahrscheinlichkeitsgrammatik erzeugt. Dabei handelt es sich um eine Grammatik, die jeder Produktion eine Wahrscheinlichkeit zuordnet. Aus dieser Grammatik kann nun ein nichtdeterministischer endlicher Automat (NFA) erzeugt werden. Dieser Automat entspricht dann dem Markov Modell. Die Ausgabe des Markov Modells ist die Summe der Einzelwahrscheinlichkeiten für jeden unterschiedlichen Weg das Wort w zu berechnen. Die Wahrscheinlichkeit für einen einzelnen Weg ist das Produkt aller Wahrscheinlichkeiten, der einzelnen Übergänge und Zustände, die durchlaufen werden um das Wort zu erzeugen.

Um nun einen möglichst gut zu den Trainingsdaten passenden Graphen zu finden wird die Formel von Bayes benutzt. Zu Beginn spiegelt das Modell genau die trainierten Daten wieder, doch nach und nach wird es flexibler indem es durch Synthese Verfahren verbessert wird.

Erkennung In der Erkennungsphase wird das vorher berechnete Markov Modell genutzt, um die Wahrscheinlichkeit für ein bestimmten Parameter zu berechnen. Da auch gültige Eingaben einen sehr kleinen Wahrscheinlichkeitswert haben können, beschränkt man die Ausgabe des Modells auf die Werte 0 und 1. Wenn das Wort mit der Grammatik erzeugt werden kann wird 1 ausgegeben sonst 0.

4.4 Token Finder

Das Token Finder Modell ist nützlich, um festzustellen, ob festgelegte Werte manipuliert wurden. Anwendungen benutzen an einigen Stellen Attribute, die in einer Liste festgelegt sind. Kann ein Benutzer in einem Formular beispielsweise eine Stadt aus dem Ruhrgebiet aus einem Drop-Down-Menü wählen, so ist vorgegeben, welche Werte in diesem Parameter übergeben werden. Beinhaltet die Liste die Städte "Dortmund", "Bochum" und "Essen", würde der Wert "München" genau so als Anomalie auffallen, wie eine beliebige andere Zeichenkette.

Lernphase In der Lernphase werden alle gültigen Werte für die Anwendung ermittelt und gespeichert.

Erkennung Während der Erkennungsphase muss lediglich überprüft werden, ob ein übermittelter Wert in der vorher angelegten Liste gültiger Werte enthalten ist oder nicht. Anomalien können mit diesem Modell direkt festgestellt werden.

4.5 Attributpräsenz

Der Schwerpunkt der nun folgenden Modelle basiert nun nicht so sehr auf einer Analyse der einzelnen Parameterinhalte, sondern zielt mehr auf die Struktur der Anfragen ab.

Programmgenerierte Queries beinhalten in der Regel immer die gleiche Anzahl an Attributen, da in der Auswertung auf alle zugegriffen wird. Die Anwendung übermittelt dadurch möglicherweise zwar auch leere Felder, deren Auswertung dann im System nicht berücksichtigt wird, jedoch sind diese auf jeden Fall in der Anfrage enthalten.

Da ein Angreifer nicht unbedingt alle Parameter kennt, die ausgewertet werden oder nur die Werte übermittelt, die ihm wichtig erscheinen, lässt sich hierdurch eine Anomalie erkennen. Sie liegt vor, wenn zu viele oder zu wenig Attribute vorhanden sind. Ein simples Beispiel für dieses Modell ist ein Login, bei dem der Angreifer nur Benutzername und Passwort übermittelt, aber übersieht, dass auch jedes mal eine Sequenznummer in der Anfrage enthalten ist. Die Nummer wird bei jedem Seitenaufruf neu generiert und ist nur einmal verwendbar.

Lernphase In der Lernphase speichert man alle Attribute, die von einer Anwendung übermittelt werden und weist ihnen einen Hashwert zu, über den sie während der Erkennungsphase ermittelt werden können. Dieser Prozess ist sehr simpel und damit auch sehr effizient.

Erkennung Wird bei der Analyse das Fehlen eines Parameters bemerkt, oder erkennt das System zu viele Werte, handelt es sich um eine Anomalie. Stimmen die vom Algorithmus überprüften Attribute mit den Hashwerten überein, ist die Anfrage normal einstuftbar.

4.6 Reihenfolge

Die Reihenfolge der Parameter innerhalb einer Anfrage ist genau wie die Attributpräsenz ein einfaches Kriterium, um zu erkennen, ob die Anfrage durch die Anwendung oder per Hand erstellt wurde.

Die Generierung innerhalb der Anwendung erfolgt immer nach dem gleichen Schema, da die Parameter nicht zufällig befüllt werden, sondern nach der Programmlogik im Quellcode sortiert sind. Wenn eine Anfrage jetzt jedoch durch einen Menschen geschieht, ordnet dieser die Werte eventuell nicht, sondern fügt sie einfach nacheinander ein. Diese menschliche Sortierung hat auf die Ausführung des eigentlich Programms zwar keinen Einfluss, fällt jedoch der Anomalieerkennung auf.

Lernphase In der Lernphase wird ein gerichteter Graph erstellt. Jeder Knoten im Graph entspricht einem Parameter aus den Anfragen. Die Reihenfolge, der Parameter entspricht der Reihenfolge der Knoten beim Durchlaufen des Graphen. Da bei der Konstruktion des Graphen theoretisch Kreise entstehen können, wird der Tarjan's Algorithmus [4] auf den Graphen angewandt. Dieser entfernt die Kreise aus dem Graphen.

Erkennung In der Erkennungsphase wird nun überprüft, ob die Reihenfolge der Attribute aus der Anfrage, welche in der Lernphase ermittelt wurde, mit der des dort erstellten gerichteten Graphen übereinstimmt. Fehlende Übereinstimmungen müssen somit durch eine Anomalie verursacht worden sein.

4.7 Zugriffshäufigkeit

Die abschließenden drei Erkennungsstrategien werten nicht mehr die Parameter aus, die der Anwendung übergeben werden, sondern arbeiten mit den anderen in den Logs enthaltenen Informationen, wie der IP Adresse, dem Zugriffszeitpunkt und dem aufgerufenen Skript.

Die Häufigkeit, mit der ein Skript von einem bestimmten Benutzer aufgerufen wird, zeigt, wie er sich innerhalb der Anwendung verhält. Diese Häufigkeit kann zwar für jeden Benutzer unterschiedlich sein, lässt sich jedoch in Mustern zusammenfassen, die für das jeweilige Skript erstellt werden. Bei einer Webanwendung, die eine Registrierung voraussetzt und dann nach einem erfolgreichen Login einen Zugang zu einer Datenbank bereitstellt, in der nach Dokumenten gesucht werden kann, finden sich verschiedene Muster für die Skripte.

Die Registrierung wird von jedem Benutzer sinnvollerweise nur einmal durchgeführt. Daher ist es auffällig, wenn ein Benutzer, der an seiner IP Adresse identifizierbar ist, mehrfach versucht die Registrierung abzuschließen, oder sich ständig neu anmeldet. Auch sollte das Registrierungsskript seltener aufgerufen werden als das Loginskript, da sich jeder Benutzer nur einmal registrieren muss, um sich in Zukunft bei jedem Besuch einloggen zu können.

Betrachtet man das Suchfeld innerhalb der Anwendung, gibt es Benutzer, die es gar nicht gebrauchen, weil sie wissen, wo sich die gewünschten Dokumente innerhalb der Menüstruktur finden lassen. Andere verwenden diese Funktion jedoch häufig und starten in kurzer Zeit mehrere Anfragen. Damit ergeben sich Muster, die besagen, dass es nicht auffällig ist, wenn ein Benutzer die Suche nie verwendet, die Suche zwar häufig benutzt wird, aber nur zu bestimmten Stoßzeiten aufgerufen wird, oder aber nur hin und wieder gesucht wird. Auffällig wären hier kontinuierliche Suchanfragen, die sehr oft auftreten und zwischen denen keine Pausen sind, in denen der Benutzer die Suchergebnisse betrachtet.

Ein weiteres Beispiel, welches sofort als Anomalie erkannt wird, ist ein Bruteforce Angriff auf den Login. Bei dieser Art des Angriff versucht der Angreifer ein Passwort herauszu-

finden, indem viele verschiedene Kombinationen getestet werden. Dadurch ergibt sich eine sehr hohe Frequenz, die auffällig ist.

Lernphase In der Lernphase wird die Zeit zwischen dem ersten und dem letzten Zugriff in gleich große Intervalle aufgeteilt. Danach wird die Anzahl der Zugriffe verschiedener Clients, bzw. IP Adressen, und die gesamten Zugriffe von allen Clients in den einzelnen Zeitabschnitten ermittelt.

Erkennung In der Erkennungsphase werden die Zugriffe innerhalb eines Zeitintervalls ermittelt. Mit Hilfe der Chebyshev Ungleichung [?] wird die Wahrscheinlichkeit für eine Anomalie berechnet. Die Summe der Zugriffe verschiedener Clients und die aller Clients im Zeitabschnitt wird addiert und danach durch zwei geteilt.

4.8 Anfrageverzögerung

Über die Log Dateien des Servers lassen sich Zeitspannen errechnen, die bei einer festen Reihenfolge von Anfragen durch einen Benutzer entstehen. Im vorherigen Abschnitt wurde eine Webanwendung beschrieben, für die eine Registrierung notwendig ist. Bei dieser Registrierung ist die Abfolge der aufgerufenen Skripte immer gleich, da sie vernetzt sind. Zuerst wählt man einen Benutzernamen und ein Passwort, dann werden auf einer weiteren Seite optionale Eingaben wie Geburtsdatum, Wohnort und Interessen abgefragt, die die anderen Benutzer als Persönlichkeitsprofil einsehen können. Als letztes bekommt der Benutzer eine Übersicht angezeigt, die er zur Vollendung der Anmeldung bestätigen muss.

Zwischen den einzelnen Seiten vergeht nun die Zeit, die benötigt wird, um Werte einzugeben. Selbst bei erfahrenen Benutzern lässt sich eine untere Zeitschranke berechnen, die auf jeden Fall gebraucht wird, wenn die minimale Anzahl an Informationen eingegeben wird. Führt jetzt kein Mensch diese Anmeldung durch, sondern ein Programm, welches speziell auf die Webanwendung abgestimmt ist, geht die Registrierung deutlich schneller und Zeitschranken werden unterschritten. Auf diese Weise wird eine Anomalie erkannt und das Programm an der automatischen Anmeldung gehindert.

Lernphase In der Lernphase werden die Zeitspannen, welche die verschiedenen Benutzer benötigen, um von einer Seite auf die Nächste zu wechseln, gespeichert. Sobald genug Trainingsdaten gesammelt wurden, bildet das System die durchschnittliche Zeit, die zum Bearbeiten einer Anfrage gebraucht wurde und hat einen Mittelwert, der mit erneuten Anfragen verglichen werden kann.

Erkennung Die Erkennung von Anomalien mit Hilfe dieses Modells ähnelt der in 4.2 beschriebenen Berechnung mittels des Pearson- X^2 -Tests [4], mit dessen Auswertung bestimmt werden kann, ob es sich um eine Anomalie handelt.

Da bei einer kleinen Anzahl von Zugriffen durch einen Benutzer nur sehr ungenaue Daten über sein Verhalten vorliegen, kann diese Strategie zu Beginn nicht stark bei der Berechnung des gesamten Anomaly Scores verwendet werden. Je mehr Zugriffe jedoch zur Analyse vorliegen, desto mehr Beachtung kann dem Wert geschenkt werden. Daher skaliert dieses Modell mit der Anzahl der Aktionen eines Benutzers.

4.9 Interaktionsreihenfolge

In einer Webanwendungen arbeiten mehrere einzelne Anwendungen zusammen und stellen im ganzen die komplette Funktionalität zur Verfügung. Bei einem Onlineshopsystem muss sich der Benutzer erst einloggen, kann dann Waren zum Warenkorb hinzufügen und dann damit zur Kasse gehen. Diese Reihenfolge ist vorbestimmt, damit der Warenkorb einem bestimmten Benutzer zugeordnet werden kann, der dann zum Schluss auch die Rechnung bekommt. Wenn ein Angreifer nun versucht direkt Waren hinzuzufügen und zur Kasse zu gehen, ohne sich vorher am System anzumelden, widerspricht dies der logischen Reihenfolge und stellt eine Verletzung der Sicherheitsregeln dar.

Die Manipulation dieser Reihenfolge dient dazu, Funktionen auszuführen, die nur nutzbar sind, wenn man mit einem Benutzernamen angemeldet und eingeloggt ist, der die entsprechenden Rechte besitzt. In einem anderen Beispiel könnte ein Angreifer versuchen einen anderen Benutzer zu löschen, ohne sich vorher als Administrator anzumelden.

Lernphase In dieser Phase werden Benutzerinteraktionen einer bestimmten Sitzung zugeordnet und lassen so Rückschlüsse auf das Bewegungsverhalten in der Anwendung zu. Die Bewegungsabläufe werden dann in einen Automaten umgewandelt, welcher alle Bewegungsmuster repräsentiert, die als gültige Eingaben aufgefasst werden.

Erkennung Solange die Aufrufe innerhalb einer Sitzung den Möglichkeiten entsprechen, die der Automat vorgibt, handelt es sich um gültige Eingaben. Versucht der Angreifer eine Aktion auszuführen, die laut Automat nicht erreichbar ist, wurde eine Anomalie erkannt.

5 Analyse

Bei dem hier vorgestellten System [3] von Christopher Kruegel und Giovanni Vigna handelt es sich um ein sehr effizientes System um Anomalien in einer Webapplikation festzustellen. Da die teils sehr aufwändigen Berechnungen nur in den Lernphasen vollzogen werden, arbeitet das Anomalieerkennungssystem in der Erkennungsphase sehr schnell. Besonders effizient arbeitet das System dank der speziellen Modelle. Je nach Wertebereich der Parameter können schon wenige Modelle alleine direkt eine Anomalie erkennen. Als Beispiel zu nennen sind die Modelle Attributlänge und Token Finder, da diese beispielsweise eine veränderte Session ID sofort erkennen können, da diese immer eine feste Länge hat, bzw. Werte die nicht in der Liste mit gültigen Token sind direkt identifizieren

können.

Auch bei der Effizienz gibt es sehr positive Ergebnisse. Die Rate der false positives war in der von Kruegel und Vigna durchgeführten Studie [3] sehr gering (< 0.000650). Da dieses System aber so konzipiert ist, dass kein Eingriff von außen notwendig sein soll, darf es keinen einzigen Fehlalarm geben. Existieren Fehlalarme, müsste man Kontrollstrukturen einführen, die im Gegensatz zur vollständigen Automatisierung ständen.

Das System operiert außerdem nur auf Zugriffslogs, was bedeutet, dass ein Angriff erst erkannt wird, wenn er schon stattgefunden hat. Daher ist das hier vorgestellte Konzept eher ein Analysesystem, welches Anomalien erkennt, sie aber nicht abzuwehren kann. Das System stellt somit keinen direkten Schutz für die analysierten Webanwendungen dar. Ein weiterer Schwachpunkt ist die Tatsache, dass bei der Analyse POST Anfragen nicht beachtet werden, da die per POST übergebenen Parameter nicht in den Zugriffslogs protokolliert werden.

6 Fazit

Das hier vorgestellte Anomalieerkennungssystem kann einen Großteil der verübten Angriffe erkennen, basiert jedoch auf Logdaten, die erst nachher aufgezeichnet werden. Wenn die Daten also zu Analyse vorliegen, wurde der Angriff bereits erfolgreich verübt.

Die Tests haben gezeigt, dass es möglich ist ein System zu entwerfen, welches nicht konfiguriert werden muss und sich daher einfach auf andere Szenarien übertragen lässt. Auffällig ist hierbei, dass trotz maschinell lernender Sicherheitsmechanismen, sehr wenig Fehlalarme auftreten, die nur mit Hilfe fester Regeln hätten verhindert werden können.

Leider schützt das System nicht vor Angriffen, die erst im Kontext der gesamten Sitzung als solche identifiziert werden könnten, da hier nur einzelne Anfragen betrachtet werden.

Um das ADS sicherer zu machen wäre jetzt ein Konzept nötig, welches die ganze Sitzung bewertet und so feststellen kann, dass alle Anfragen leicht von der Norm abgewichen sind und dadurch als gesamtes eine Anomalie erzeugen. Jedoch würde dieses Konzept den Rahmen dieser Arbeit sprengen.

Das zeigt, dass die vorgestellten Erkennungsmodelle zwar einen sehr guten Schutz bilden, der sich leicht auf andere Systeme übertragen und überall dort nutzen lässt, wo Parameter analysiert werden, dies aber nicht ausreichend ist, um eine komplette Anwendung zu überwachen.

Literatur

- [1] BILLINGSLEY, P.: *Probability and Measure, third ed.* Wiley-Interscience, 1995.
- [2] HAYTER, ANTHONY: *Probability and Statistics for Engineers and Scientists, second ed.* DuxburyPress, 2001.
- [3] KRUEGEL, C., G. VIGNA und W. ROBERTSON: *A multi-model approach to the detection of web-based attacks.* Computer Networks, 48(5):717–738, 2005.
- [4] TARJAN, R.: *Depth-first search and linear graph algorithms, SIAM Journal of Computing 1 (2).* 1972.