MASTERARBEIT

WEB APPLICATION SECURITY

Application Firewalls

ausgeführt am

Studiengang

Informationstechnologien und Wirtschaftsinformatik

Von: Maximilian Steiner

Personenkennzeichen: 1910320020

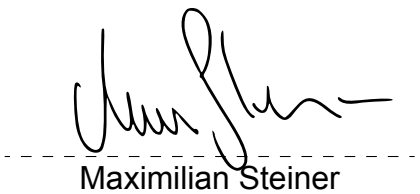
Graz, am 28. Juni 2022

.....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.



Maximilian Steiner

DANKSAGUNG

Hiermit möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Masterarbeit begleitet und unterstützt haben.

Dank gilt meiner gesamten Familie und all meinen Freunden. Ihr habt laufend dafür gesorgt, mir die notwendige Motivation zu geben, um in den letzten Zügen meines Abschlusses nicht den Kopf zu verlieren und habt mich an motivationslosen Tagen aufgeheitert und zum Schreiben animiert.

Dank gilt außerdem meinem Chef Ing. Thomas Bauer. Du hast mir nicht nur die Möglichkeit gegeben, neben meinem Job diesen Weg einzuschlagen, sondern mir auch in dunklen Zeiten deine volle Unterstützung zugesagt und immer aufbauende Worte gefunden.

Darüber hinaus gebührt DI Markus Petelinc besonderer Dank für seine Betreuung im Zuge der Ausarbeitung. Sie haben mir stets konstruktives Feedback gegeben und mit Ihren wertvollen Denkanstößen dafür gesorgt, das Wesentliche im Auge zu behalten und schlussendlich eine gute Arbeit zu verfassen.

KURZFASSUNG

In der heutigen Zeit spielt die Absicherung von modernen IT-Systemen eine wichtige Rolle und stellt gleichzeitig eine Herausforderung für viele Unternehmen dar. Bei den Versuchen, strikte Richtlinien umzusetzen und die eigenen Daten zu schützen, stoßen einige davon an ihre Grenzen. Die konstant wachsende Anzahl an modernen Bedrohungen forciert dabei hohe Investitionen und regelmäßige Überprüfung, Anpassung und Verbesserung der unternehmensinternen Prozesse. Aufgrund der Diversität der Bedrohungen stellen unterschiedliche Systeme dabei unterschiedliche Funktionalitäten bereit und es gibt keine zentrale Stelle für Abwehrmechanismen. Dabei stellt sich die Frage, ob eine zentrale Abhandlung diverser Bedrohungen auf Seiten einer Applikation sinnvoll ist, oder ob davon abgesehen werden sollte.

Um dies zu beantworten werden sicherheitsrelevante Aspekte im Bereich von Web Application Security untersucht und ein Einblick in die Relevanz zur Absicherung von Web Applikationen gegeben. Darüber hinaus werden einige der gängigsten Bedrohungen von modernen Systemen analysiert und beschrieben. Neben Konzepten, die dabei unterstützen, die Sicherheit von Software-Systemen zu erhöhen, werden ebenfalls die Möglichkeiten zur Auslagerung sicherheitsrelevanter Aspekte untersucht.

Der Fokus der Arbeit bezieht sich auf Web Application Firewalls und es wird konkret auf deren Verwendung, Architektur und deren Grenzen eingegangen. In diesem Kontext werden sowohl Lösungen kommerzieller Anbieter, als auch bekannte Open-Source Lösungen vorgestellt und verglichen. Die Open-Source Lösung ModSecurity wurde dabei für eine Anwendung in der Praxis ausgewählt. Es wurde ein Konzept aufgestellt, um die Integrität von HTTP-Cookies zu validieren und dieses wurde mithilfe eines eigens erstellten Regelwerks innerhalb von ModSecurity umgesetzt. Die Umsetzung dieses Konzepts hat gezeigt, dass eine solche moderne Anforderung auch erfüllt werden kann, ohne dabei die betroffene Applikation an sich verändern zu müssen.

Anhand der zuvor durchgeführten Recherche und der praktischen Anwendung wird einerseits die Flexibilität und der Nutzen einer WAF verdeutlicht, und andererseits dargelegt, dass eine eigens adaptierte Implementierung keinen Mehrwert gegenüber der Verwendung bestehender Lösungen bietet.

ABSTRACT

In today's world, securing modern IT systems plays an important role, while also posing a challenge for many companies. In their attempts to implement strict policies and protect their own data, some of them come up against their limits. The constantly growing number of modern threats forces high investments and regular review, adaptation and improvement of the company's internal processes. Due to the diversity of the threats, different systems provide different functionalities, and there is no central point for defense mechanisms. This raises the question of whether or not it makes sense to deal with diverse threats centrally within an application.

To answer this question, this thesis examines the security-relevant aspects in the area of web application security and then illustrates their relevance for securing web applications. Furthermore, some of the most common threats to modern systems are analyzed and described. In addition to concepts that help to increase the security of software systems, the possibilities for outsourcing security-relevant aspects are also examined.

The main part of this thesis then introduces Web Application Firewalls in terms of their use, architecture and limits and compares both commercial and well-known open-source solutions. The open-source solution ModSecurity was then selected for a practical illustration of how to secure a web application. A concept for validating the integrity of HTTP-Cookies was created, and it was shown that such a modern requirement can be fulfilled by using a custom ruleset within ModSecurity, without needing to apply changes to the application itself.

The results from both the literature research and the practical application highlight the flexibility and benefits of Web Application Firewalls and demonstrate that an adapted implementation ultimately offers no added value compared to the use of existing solutions.

INHALTSVERZEICHNIS

1 Einleitung **1**

1.1 Aufgabenstellung 1
1.2 Zielsetzung der Arbeit 2
1.3 Vorgehen und Methodik 3
1.4 Aufbau der Arbeit 3

2 Webapplication Security **4**

2.1 Gründe, Ziele und Bedrohungen 4
2.1.1 Gründe und Ziele 5
2.1.2 Bedrohungen 11
2.2 Konzepte und Patterns 22
2.2.1 Design Eigenschaften 23
2.2.2 Minimale Offenlegung 24
2.2.3 Starke Durchsetzung 28
2.2.4 Redundanz 28
2.2.5 Vertrauen & Verantwortung 29
2.3 Auslagerungsmöglichkeiten 30
2.3.1 Kriterien 30
2.3.2 Managed Security Service Provider 31
2.3.3 Managed Detection and Response Provider 32

3 Web Application Firewalls **34**

3.1 Anwendungsfälle und Verwendung 38
3.1.1 Schutz vor SQL-Injection 40
3.1.2 Schutz vor XSS 41
3.1.3 Schutz vor Session Tampering 41
3.2 Architektur 43
3.2.1 Sicherheitsmodelle & Regelwerke 43
3.2.2 Typen & Betriebsarten 48
3.3 Grenzen & Herausforderungen 55
3.3.1 Herausforderungen 56
3.3.2 Grenzen 58

3.4 Anbieter 59
3.4.1 Kommerziell 59
3.4.2 Open Source 65
3.4.3 Gegenüberstellung 69

4 Cookie Validierung mit ModSecurity **71**
4.1 Konzept 72
4.2 Test-Szenario 74
4.2.1 Web-Applikation 74
4.2.2 ModSecurity Modul 76
4.3 Praktische Umsetzung 79

5 Kritische Reflexion **91**

ABBILDUNGSVERZEICHNIS **93**

TABELLENVERZEICHNIS **94**

LISTINGS **95**

LITERATURVERZEICHNIS **96**

1 Einleitung

In Zeiten von Cloud-Computing, Software as a Service (SaaS) und der fortlaufenden Vernetzung unterschiedlichster Dienstleistungen über den öffentlichen Raum des World Wide Web, ist die Sicherheit, beziehungsweise die Absicherung, von einzelnen Services oder auch ganzen Micro-Service-Architekturen ein Thema, das nicht nur die dienstleistenden Unternehmen betrifft, sondern ebenfalls die Nutzer- und AnwenderInnen dieser Services selbst (Leenes, van Brakel, Gutwirth & De Hert, 2017).

In dieser digitalisierten Welt steigt ebenfalls die Anzahl an sensiblen unternehmens- und personenbezogenen Daten auf ein unvorstellbares Ausmaß. Smartphones haben mittlerweile mehr Rechenleistung als die ersten Computer und konsumieren und produzieren Daten im entsprechenden Ausmaß.(Bentz, 2019)

Durch die stetig wachsende Menge dieser sensiblen Informationen findet ebenfalls ein Umdenken bezüglich des Schutzes dieser Daten statt, das beispielsweise zur Entwicklung und Umsetzung der Datenschutzgrundverordnung (DSGVO) führte, die den Schutz von personenbezogenen Daten in der Europäischen Union vorsieht (Voigt & von dem Bussche, 2017).

1.1 Aufgabenstellung

Unternehmen stehen demnach vor der Herausforderung, sich selbst, wie auch alle gesammelten und verarbeiteten Daten ihrer Nutzerinnen und Nutzer vor der Außenwelt zu schützen. Dazu müssen sowohl die eigenen Fähigkeiten verstanden werden als auch die Grenzen der angebotenen Services und verwendeten Sicherheitstechnologien. Netzwerk Firewalls oder auch Intrusion Prevention Systeme können dabei verwendet werden, um eine große Menge von Bedrohungen auf niedriger Ebene zu filtern, stoßen aber bei der zunehmenden Anzahl an anwendungsspezifischen Bedrohungen an ihre Grenzen (Nayak, Ray & Ravichandran, 2022).

Diese Arbeit untersucht sicherheitsrelevante Aspekte von Webapplikationen und legt einen Fokus auf Web Application Firewalls (WAF), welche zur gezielten Filterung, Blockierung und auch Überwachung von ein- und ausgehendem HTTP-Traffic von Web-

Applikationen verwendet werden kann (Dermann et al., 2008).

Aus dem Thema der Arbeit und den ersten Recherchen ergibt sich folgende Forschungsfrage für diese Arbeit:

- Inwieweit ist es sinnvoll und möglich, das Blockieren anwendungsspezifischer Bedrohungen, wie es von Web Application Firewalls gemacht wird, durch Adaption bestehender Konzepte von Web Application Firewalls applikationsseitig zu implementieren?

In diesem Zusammenhang ergeben sich zudem die folgenden möglichen Hypothesen:

- Hypothese 1: Eine eigene adaptive Implementierung bringt einen Mehrwert gegenüber der Verwendung existierender Lösungen.
- Hypothese 2: Existierende Lösungen bringen einen Mehrwert gegenüber einer eigenen adaptiven Implementierung.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, die Gründe und die Bedeutung der Absicherung von Webapplikationen in der heutigen Zeit darzustellen. Neben der Behandlung von grundlegenden Konzepten und Patterns zur Sicherung von Webapplikationen, sowie Recherchen zur Auslagerung sicherheitsrelevanter Aspekte, wird ein Fokus auf Web Application Firewalls gelegt.

Dabei wird einerseits dargelegt, wie sich WAF von regulären Firewalls abgrenzen, und andererseits, gegen welche Risiken eine Applikation oder ein Service mithilfe von Web Application Firewalls geschützt werden kann, wie beispielsweise Cross-Site-Scripting (XSS) oder SQL-Injection (Dermann et al., 2008). Die Arbeit legt nicht nur die Grundlagen von WAF dar, sondern analysiert und beschreibt darüber hinaus bereits bestehende Systeme und zeigt die Umsetzung einer modernen Anforderung anhand eines realitätsnahen Beispiels in der Praxis.

1.3 Vorgehen und Methodik

Im ersten Schritt wird durch Literaturrecherche die Notwendigkeit der Absicherung von Webapplikationen in der heutigen Zeit dargestellt. Es werden sowohl bekannte Methoden und Konzepte als auch häufig auftretenden Risiken beziehungsweise Bedrohungen analysiert und beschrieben.

In weiterer Folge wird ein Einblick in Web Application Firewalls gegeben. Dabei werden neben den tatsächlichen Anwendungsfällen auch die Grenzen von WAF eruiert und beschrieben. Darüber hinaus werden sowohl Open-Source Lösungen als auch Systeme von bekannten kommerziellen WAF-Anbietern untersucht. Schlussendlich wird die Möglichkeit zur Implementierung einer modernen Anforderung mithilfe einer WAF anhand eines praktischen Beispiels realitätsnah dargelegt.

1.4 Aufbau der Arbeit

Einleitend werden grundlegende Aspekte für Security im Bereich von Web-Applikationen dargelegt. Hierbei werden einerseits Gründe für die Absicherung von Webapplikationen untersucht und bekannte Bedrohungen der heutigen Zeit für solche Applikationen beschrieben.

Anschließend werden bekannte Konzepte, Patterns und Auslagerungsmöglichkeiten, die für den Schutz und die Absicherung von Webapplikationen in der heutigen Zeit zum Einsatz kommen, vorgestellt und im Detail beschrieben.

In weiterer Folge wird der Fokus auf Web Application Firewalls gelegt. Neben konkreten Anwendungsfällen stehen hierbei ebenfalls die grundsätzliche Architektur, Grenzen von WAF als auch bekannte kommerzielle und Open-Source Anbieter im Vordergrund.

Schlussendlich wird das Konzept von Web-Applikation Firewalls anhand eines praxisnahen Beispiels dargestellt. Es werden die Anforderungen an eine derartige System-Architektur analysiert als auch eine praktische Umsetzung durchgeführt.

2 Webapplication Security

Online zu sein ist heute keine Besonderheit mehr, sondern etwas, das uns im alltäglichen Leben ständig und überall begleitet. Vom Computer, über das Mobiltelefon bis hin zum Kühlschrank verbinden sich unzählige Geräte in das Internet of Things (IoT), um mit zahlreichen anderen Geräten und Dienstleistungen zu kommunizieren und Informationen auszutauschen. Dadurch werden Umgebungen geschaffen, die die Art und Weise wie wir Geschäfte tätigen, kommunizieren und leben wesentlich intelligenter und autark machen (Kolokotronis & Shiaeles, 2021).

Die zunehmende Anzahl an Branchen und Sektoren, die von der technologischen Revolution erfasst werden, erweitern das Angebot und die Möglichkeit des Konsums von Informationen und Dienstleistungen konstant. Damit einher erhöht sich ebenfalls die Menge und Art der Daten, die ausgetauscht, verarbeitet und gespeichert werden. Viele Unternehmen nutzen die gesammelten Daten, um ihren Nutzerinnen und Nutzern eine auf sie zugeschnittene Erfahrung zu bieten. Andere stellen Möglichkeiten bereit, administrative Tätigkeiten des Alltags jederzeit und ortsunabhängig zu erledigen. Dabei spielen vor allem sensible, personenbezogene Daten eine große Rolle (Kolokotronis & Shiaeles, 2021).

Diese Daten beinhalten oft sensible Informationen, die sich auf reale Personen, Unternehmen oder Transaktionen beziehen. Die Sicherheit solcher Daten sollte für jedes Unternehmen oberste Priorität haben und wird durch gesetzliche Richtlinien, wie der Datenschutzgrundverordnung, vorausgesetzt (Leenes et al., 2017).

In diesem Kapitel werden die Gründe zum Schutz dieser Daten analysiert. Es wird ein Einblick in bekannte Bedrohungen gegeben und bewährte Verfahren beschrieben, um vor solchen Risiken geschützt zu sein. Darüber hinaus werden Möglichkeiten untersucht, um sicherheitsrelevante Aspekte auslagern zu können.

2.1 Gründe, Ziele und Bedrohungen

Die Tiefen des Internets geben seinen Nutzerinnen und Nutzern ein Gefühl von Anonymität und stellen ihnen unzählige Dienstleistungen und Informationen kostenlos

oder kostenpflichtig zur Verfügung. Hinter den Anwenderinnen und Anwendern steckt allerdings weit mehr als eine IP-Adresse. Von Anonymität kann heute nicht mehr gesprochen werden. Das Sammeln von nutzerspezifischen Daten nimmt immer mehr Raum ein, gerade bei vermeintlichen kostenlosen Dienstleistungen. Die kommerzielle Verwertung dieser Daten hat sich inzwischen zu einem eigenen Geschäftsmodell entwickelt und der Beitrag, den einzelne Nutzerinnen und Nutzer dazu leisten, ist weitgehend intransparent (Schaar, 2020).

Daten spielen mittlerweile eine Schlüsselrolle in der globalen digitalen Wirtschaft und bilden das Fundament für die Monetarisierung von scheinbar kostenlosen Dienstleistungen. Daten sind grundsätzlich zu einem wertvollen Gut geworden und werden als Währung der Zukunft bezeichnet. Die Verarbeitung dieser Daten findet speziell im Bereich von wirtschaftlichen und sozialen Aktivitäten statt (Voigt & von dem Bussche, 2017).

Die Europäische Union hat bereits im Jahr 1995 eine erste Richtlinie zum Schutz von Daten aufgestellt und diese mit der Einführung der Datenschutzgrundverordnung im Jahr 2016 über die europäischen Grenzen hinaus erweitert. Ziel der DSGVO ist es, das Vertrauen der Menschen in einen verantwortungsvollen Umgang mit ihren personenbezogenen Daten zurückzugewinnen und Unternehmen anzuleiten, ihren Datenschutzverpflichtungen nachzukommen (Voigt & von dem Bussche, 2017).

Warum der Schutz dieser Daten so wichtig ist und welche Risiken bestehen, wird in den folgenden Abschnitten erläutert.

2.1.1 Gründe und Ziele

Unabhängig davon welche Dienstleistung ein Unternehmen anbietet, ob diese Dienstleistung für die Nutzerinnen und Nutzer nun kostenlos oder kostenpflichtig ist oder ein Service nur für interne Zwecke benötigt wird, wird für die Bereitstellung dieser Dienstleistung Wissen (Know-How) aufgebaut und konsumiert (Leenes et al., 2017).

Dieses Know-how ist sowohl für das Unternehmen selbst, als auch für dessen Kunden von großer Bedeutung und muss entsprechend abgesichert werden. Ist das nicht der Fall, droht dem Unternehmen gegebenenfalls ein wirtschaftlicher Schaden und sensible Kunden- und Unternehmensdaten könnten von Dritten missbraucht werden (Leenes et al., 2017).

Um das zu vermeiden, wurden in der Informationstechnologie im Laufe der Zeit soge-

nannte Schutzziele definiert, welche sich in die Oberbegriffe **Vertraulichkeit**, **Integrität** und **Verfügbarkeit** gliedern lassen. Diese Schutzziele wurden mit der Absicht definiert, gespeicherte und zu verarbeitende Informationen vor dem Missbrauch Dritter zu schützen. In den 90er Jahren wurde Vertraulichkeit noch annähernd mit Sicherheit gleichgesetzt, mittlerweile wird dieses Ziel allerdings von Integrität und Verfügbarkeit begleitet, welche sich wiederum in Unterziele gliedern lassen (Bedner & Ackermann, 2010).

Vertraulichkeit

Vertraulichkeit (confidentiality) ist das erste Schutzziel, das in dieser Arbeit genauer betrachtet wird. Vertraulichkeit eines IT-Systems ist dann gegeben, wenn die enthaltenen Informationen nur für Personen und Services zur Verfügung stehen, die auch die Befugnis haben diese zu konsumieren. Um dieses Ziel erreichen zu können, sind Maßnahmen erforderlich, die die zulässigen Informationsflüsse zwischen den konsumierenden Beteiligten des Systems definieren als auch kontrollieren, wie beispielsweise Zugriffsrechte und Zugriffsschutz (Bedner & Ackermann, 2010).

Vertraulichkeit bezieht sich allerdings nicht nur auf persistente Daten, sondern ebenfalls auf den Transport beziehungsweise den Austausch dieser. Demnach muss neben dem eigentlichen Inhalt der Informationen (Schutz der Informationsinhalte) ebenfalls das Verhalten der konsumierenden Individuen oder Systeme vor Beobachtung, Aufzeichnung und Auswertung durch Dritte geschützt werden (Schutz des Informationsverhaltens). Um dies zu gewährleisten setzt man unter anderem auf einen geeigneten Zugriffsschutz als auch auf eine angemessene Verschlüsselung der Informationen (Bedner & Ackermann, 2010).

Neben den genannten Aspekten schließt die Vertraulichkeit ebenfalls noch folgende Unterziele mit ein (Bedner & Ackermann, 2010):

- Unverkettbarkeit
- Nicht-Verfolgbarkeit
- Unbeobachtbarkeit
- Verdecktheit
- Anonymität
- Pseudonymität

Unverkettbarkeit (unlinkability) hat zum Ziel, dass mehrere verkettete Informationsaufrufe nicht miteinander in Verbindung gebracht werden können und kann erreicht werden, indem der Informationsaustausch über mehrere Server geleitet wird. Der Austausch der Daten erfolgt demnach nicht direkt vom Sender zum Empfänger, sondern wird über mehrere Zwischenstationen, sogenannte „Mix-Server“, geleitet, wodurch die Zuordnung der einzelnen Datenpakete zu einem bestimmten Nutzer verschleiert wird (Bedner & Ackermann, 2010). Laut Wendzel (2018), kann ein solcher Mix-Server dabei mit einem Router verglichen werden. Sie dienen in erster Linie dazu, um Informationen weiterzuleiten, erweitern diese Funktionalität allerdings um asymmetrische Kryptografie und stellen so sicher, dass die Kommunikationsbeziehung zwischen Sender und Empfänger anonymisiert wird (Wendzel, 2018).

Nicht-Verfolgbarkeit (untraceability) bezieht sich darauf, dass es nicht möglich sein darf, Aktionen oder Inhalte zu einer bestimmten Person zu identifizieren oder nachvollziehen zu können (Bedner & Ackermann, 2010).

Unbeobachtbarkeit (unobservability) hat den Zweck, Sender und Empfänger beim Austausch von Informationen zu verschleiern. Es darf demnach nicht erkannt werden, wer Daten sendet oder empfängt. Dieses Schutzziel kann allerdings bereits erreicht werden, wenn der anfangs beschriebene Schutz des Informationsverhaltens erfüllt ist und es Dritten nicht möglich ist, das System zu nutzen oder Nachrichten über das System zu senden (Bedner & Ackermann, 2010).

Die **Verdecktheit (covertness, obscurity)** setzt an der Unbeobachtbarkeit an und soll dafür sorgen, dass es für Dritte nicht möglich ist zu erkennen, dass überhaupt Daten ausgetauscht werden. Demnach wissen nur die kommunizierenden Parteien wann, wie und wo Daten ausgetauscht werden. Als reales Beispiel kann Steganografie genannt werden. Dabei werden Informationen im Rauschen von Musik- oder Bilddateien versteckt (Bedner & Ackermann, 2010).

Anonymität (anonymity) und Pseudonymität (pseudonymity) sind eng miteinander verbunden. Anonymität bedeutet, dass jemand innerhalb einer Menge nicht eindeutig identifizierbar ist. Je größer diese Menge ist, desto größer ist ebenfalls die Anonymität. Pseudonymität bezeichnet eine abgeschwächte Form der Anonymität. Dabei werden anstatt realer Identitäten Pseudonyme verwendet, um die tatsächliche Identität zu verschleiern. Die Verwendung von Pseudonymen kann im Umkehrschluss allerdings wiederum die Identität preisgeben (Wendzel, 2018).

Neben den genannten Aspekten des Schutzzieles „Vertraulichkeit“ gibt es zu dieser

laut Bedner (2010) auch ein Gegenstück. Und zwar die **Transparenz (transparency)**. Dieses Schutzziel zielt auf die Nachvollziehbarkeit, Klarheit und Erkennbarkeit von Systemen ab und soll deren Arbeits- und Funktionsweise durchschaubar machen. Um Transparenz in einem System zu gewährleisten, können Instrumente wie Audits, Protokolle und Log-Dateien oder Code Reviews verwendet werden. Transparenz selbst lässt sich wiederum in **Zurechenbarkeit (accountability)**, **Authentizität (authenticity)** und **Revisionsfähigkeit (reviewability)** gliedern (Bedner & Ackermann, 2010).

Ein System ist zurechenbar, wenn Aktionen oder Dokumente dem jeweiligen Urheber zugeordnet werden kann. Es muss demnach zu jedem Zeitpunkt klar sein, welcher Person oder welchem System eine ausgeführte Aktion zuzuordnen ist und wodurch diese ausgelöst wurde. Zurechenbarkeit bietet das Gegenstück zur Nicht-Verfolgbarkeit und kann beispielsweise durch den Einsatz von Protokollen, Zeitstempeln und digitalen Signaturen gewährleistet werden (Bedner & Ackermann, 2010).

Authentizität hingegen gewährleistet, dass Informationen von der angegebenen Quelle stammen und dass die Identität des Herausgebers dieser Information über die gesamte Kommunikationskette hinweg korrekt und vertrauenswürdig ist. Dies kann beispielsweise über Authentisierung, Authentifizierung und Autorisierung mithilfe von Credentials[1] geschehen. Eine Person oder ein System authentisiert sich erfolgreich an einem anderen System. Dieses gleicht dessen Credentials ab und authentifiziert die Person oder das System für den Zugriff. Danach wird der Zugriff auf Ressourcen basierend auf Gruppen oder Rollen vom System autorisiert (Bedner & Ackermann, 2010).

Die Revisionsfähigkeit bezieht sich auf die Nachvollziehbarkeit und Prüfbarkeit von Handlungen oder Aktivitäten innerhalb eines Systems. Diese kann durch Protokollierung und Dokumentation gewährleistet werden und soll aufdecken, wer, wann, welche Daten und in welcher Weise verarbeitet hat. Es muss demnach einerseits ersichtlich sein, wer Daten in ein System geschrieben oder daraus entfernt hat und andererseits welche Änderungen diese Daten erfahren haben (Bedner & Ackermann, 2010).

Die Revisionsfähigkeit ist ähnlich der Zurechenbarkeit, lässt sich davon allerdings dahingehend abgrenzen, dass die Zurechenbarkeit sich auf aktuelle Handlungen von Personen fokussiert und die Revisionsfähigkeit sich eher auf das System und dessen nachträgliche Überprüfbarkeit bezieht (Bedner & Ackermann, 2010).

[1] Eindeutiges Erkennungsmerkmal eines Nutzers oder Systems.

Integrität

Integrität (integrity) bezieht sich einerseits auf Daten beziehungsweise Informationen an sich und andererseits auf die Systeme, welche diese Daten bereitstellen. Man unterscheidet daher folgende Arten der Integrität (Bedner & Ackermann, 2010):

- Datenintegrität
- Systemintegrität

Unter **Datenintegrität** versteht man die Unversehrtheit beziehungsweise Vollständigkeit und Korrektheit der Informationen. Sind alle Teile der Information vorhanden, spricht man von Vollständigkeit. Korrekt sind Daten, wenn sie den Kontext den sie widerspiegeln unverfälscht und ohne Manipulation wiedergeben (Bedner & Ackermann, 2010).

Bei der **Systemintegrität** spricht man von der korrekten Funktionsweise des Systems, das diese Daten bereitstellt. Die Informationen dürfen weder vom System, das sie bereitstellt, noch von Dritten manipuliert worden sein, um Integrität zu erreichen (Bedner & Ackermann, 2010).

Sollte dies doch der Fall sein, müssen Techniken existieren, die die Manipulationen entsprechend erkennen und dokumentieren. Dies hat den Vorteil, dass unsachgemäß geänderte Daten vor der Verarbeitung erkannt werden und so die Entstehung von etwaigen Risiken minimiert werden kann (Bedner & Ackermann, 2010).

Falls es technisch realisierbar ist, ist es ebenfalls wünschenswert, dass fälschlich manipulierte Daten wiederhergestellt werden können. Je nach System kann dies allerdings nicht immer gewährleistet werden (Bedner & Ackermann, 2010).

Abgesehen von Daten- und Systemintegrität umschließt die Integrität ebenso noch folgende Schutzziele (Bedner & Ackermann, 2010):

- Verlässlichkeit
- Beherrschbarkeit
- Nicht-Vermehrbarkeit

Unter **Verlässlichkeit (dependability, reliability)** versteht man ein System, welches abgesehen von seinem erwarteten Zustand keine anderen Zustände annehmen kann und seine erwartete Funktion zuverlässig erbringt. Diese Aspekte werden ebenfalls als

Funktionssicherheit oder Ordnungsmäßigkeit bezeichnet. Ein Beispiel um die Verlässlichkeit eines Systems zu steigern oder zu prüfen sind beispielsweise Unit-Tests in der Softwareentwicklung (Bedner & Ackermann, 2010). Dabei werden einzelne Komponenten eines Systems und deren Funktionsweise mit beispielhaften Daten simuliert, um Abweichungen und Verbesserungen zu erkennen, welche nicht Teil der eigentlichen Funktionalität sind (Bedner & Ackermann, 2010).

Die **Beherrschbarkeit (controllability)** bezieht sich darauf, dass ein System kein Eigenleben entwickelt und frei von Seiteneffekten ist. Die Nutzerinnen und Nutzer des Systems werden vor unerwünschten Auswirkungen durch den Einsatz des Systems geschützt. Dies kann einerseits durch die Validierung von Eingabedaten erreicht werden und andererseits durch frühzeitiges Erkennen und Abfangen von ungültigen Daten (Bedner & Ackermann, 2010).

Die **Nicht-Vermehrbarkeit (non-propagation)** gewährleistet, dass Unberechtigte nicht dazu in der Lage sind, Informationen zu kopieren und diese in weiterer Folge gezielt für Angriffe mit einer vorgetäuschten Identität zu nutzen. Solche Angriffe werden Replay-Angriffe genannt und können durch den Einsatz von sogenannten Nonces[2] verhindert beziehungsweise erschwert werden (Bedner & Ackermann, 2010).

Das Schutzziel der Integrität erlaubt es uns demnach festzustellen, dass gewisse Sachverhalte so sind wie sie sind. Im Gegensatz dazu erwähnt Bedner (2010) außerdem noch das Schutzziel „Kontingenz“, welches uns hingegen die Möglichkeit bieten soll, festzustellen, dass ein Sachverhalt anders sein könnte, als er tatsächlich ist. Kontingenz hat den Nutzen, sich nicht von der Technik einengen zu lassen und soll die Möglichkeit zur Intervention bieten (Bedner & Ackermann, 2010).

Verfügbarkeit

Das Schutzziel **Verfügbarkeit** hat den Zweck, sicherzustellen, dass jegliche Systeme zu jeder Zeit verfügbar sind und die benötigten Informationen bereitstellen können. Das setzt voraus, dass einerseits alle notwendigen Informationen inhaltlich korrekt sind und zeitnah zur Verfügung stehen und andererseits, dass die Verarbeitung dieser Informationen ordnungsgemäß funktioniert (Bedner & Ackermann, 2010).

Die Verfügbarkeit von Systemen ist oft vertraglich anhand von Service-Level-Agreements (SLA) geregelt und kann gemessen werden. Kann ein System einer vertraglich geregelten Verfügbarkeit beispielsweise nicht nachkommen, kann das anbietende Unternehmen

[2] Eine zufällige, einmalig gültige Zeichenketten (Bedner & Ackermann, 2010).

des Systems geklagt werden. Erstrebenswert ist eine Verfügbarkeit von 100% (Bedner & Ackermann, 2010).

Verfügbarkeit kann beispielsweise durch Redundanz erreicht werden. Einzelne Services, ganze Server oder sogar komplette Rechenzentren können dazu „gespiegelt“ werden und im Falle eines Ausfalles einspringen oder grundsätzlich zur Lastenverteilung genutzt werden (Wendzel, 2018).

Verfügbarkeit lässt sich ebenfalls durch Heterogenität steigern. So kann ein System beispielsweise auf unterschiedliche Hard- und Softwarekomponenten setzen, um einen Service für unterschiedliche Betriebssysteme bereitzustellen, wie es beispielsweise bei DNS der Fall ist (Wendzel, 2018).

Die Verfügbarkeit ist allerdings nicht nur direkt vom jeweiligen System abhängig, sondern von einigen weiteren Faktoren, wie Beispielsweise den Räumlichkeiten in denen sich die Kernkomponenten eines Systems befinden. Um eine hohe Verfügbarkeit zu erreichen, müssen ebenfalls diese abgesichert werden, seien es Brandschutzvorrichtungen oder Notstromaggregate (Wendzel, 2018).

2.1.2 Bedrohungen

Aufgrund großer Fortschritte im Bereich der Digitalisierung gibt es gerade auf technischer Ebene eine Vielzahl an Bedrohungen. Applikationen kommunizieren über unterschiedliche Protokolle und Formate miteinander. Einige dieser Protokolle sind mit ihren Sicherheitsdefiziten bereits tief in den Genen des Internets verankert, wie beispielsweise das TCP/IP Protokoll, Domain Name System (DNS) oder das Border Gateway Protocol (BGP). Ohne zusätzliche Sicherheitsmaßnahmen kann deren Inhalt nicht nur gelesen oder manipuliert werden, sondern es können auch Aktivitäten von Nutzerinnen und Nutzern abgegriffen werden (Schaar, 2020).

Moderne Web-Applikationen basieren auf weit mehr als den genannten Protokollen. Die konstante Weiterentwicklung und Verbesserung von alter sowie die Einfuhr neuer Technologien ergibt neben neuen Möglichkeiten ebenfalls neue Sicherheitslücken (Schaar, 2020).

Um das Spektrum an Bedrohungen genauer betrachten zu können, ist es zunächst notwendig zu verstehen, wie moderne Web-Applikationen aufgebaut sind.

Moderne Web-Applikationen

In den Anfängen erfolgte die Übermittlung und Bereitstellung von Daten durch einen einfachen Austausch von HTML, CSS und später auch JavaScript-Dateien. Dabei sendet der Client eine Anfrage zu einer serverseitigen Applikation, welche die Anfrage direkt mit den entsprechenden Daten beantwortet. Für jede weitere Ressource muss erneut eine HTTP-Anfrage an die gleiche Applikation gesendet werden. Dieses Design, in welchem eine einzelne Applikation alle Daten selbst bereitstellt, wird auch als Monolith bezeichnet (Hoffman, 2020).

Im Gegensatz zur veralteten monolithischen Struktur bilden moderne Web-Applikationen meist eine Kombination mehrerer einzelner Applikationen, die gemeinsam als geschlossenes System arbeiten und über Netzwerk-Protokolle kommunizieren. Solch ein Design wird als Microservice-Architektur bezeichnet. Jede Komponente eines solchen Systems hat eine spezielle Aufgabe und die Kommunikation untereinander findet meistens über sogenannte REST-Schnittstellen statt (Hoffman, 2020). Abbildung 2.1 zeigt den strukturellen Unterschied zwischen dem monolithischen Ansatz und einer Microservice-Architektur.

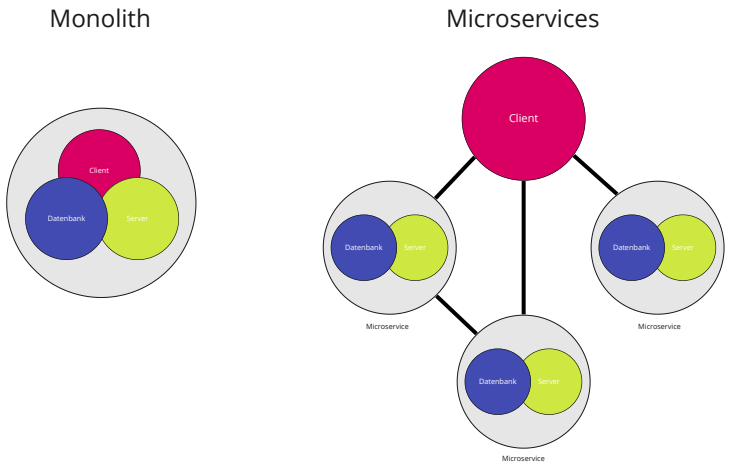


Abbildung 2.1: Monolith vs. Microservice Design[3]

REST steht hierbei für *Representational State Transfer* und bezeichnet ein Konzept zum Austausch von Ressourcen mit einem gewissen Zustand. Einen Service, der auf diesem Konzept beruht, bezeichnet man als *Restful* und dieser definiert sich über folgende Eigenschaften (Subramanian & Raj, 2019):

- Folgt der **Client-Server Architektur**: Ein oder mehrere Clients fragen einen Server nach bestimmten Ressourcen. Der Server ist zuständig dafür, die Resource zu

[3]Grafik adaptiert von (Omelchenko, 2021).

finden und bereitzustellen.

- Ist **zustandslos**: Jede Anfrage von Client oder Server enthält alle notwendigen Informationen um die Anfrage bearbeiten zu können. Zwischen zwei Anfragen müssen keine Daten zwischengespeichert werden

- **Ermöglicht Caching**: Daten, die oft angefragt werden aber sich nicht laufend ändern, können zwischengespeichert werden, um Abfragen zu beschleunigen.

- **Einheitliche Schnittstelle**: Bietet ein vordefiniertes Vokabular und stellt eine einheitliche Schnittstelle zum Abfragen der Daten bereit. Meistens wird das HTTP-Protokoll verwendet. Dieses bietet beispielsweise die Methoden GET, POST, PUT oder DELETE zur Abfrage, Manipulation oder Löschen von Daten.

- **Mehrschichten-System**: Die Systemarchitektur ist in mehrere Schichten gegliedert, von denen jede Schicht eine eigene Aufgabe besitzt und nur mit der darüber oder darunterliegenden Schicht kommunizieren kann. Beispielsweise kann ein derartiger Service eine Schicht haben die für Authentifizierung zuständig ist, eine weitere zur Lastenverteilung und eine für den eigentlichen Zugriff auf die Daten. Zwar bringt ein solches Schichtensystem zusätzliche Latenzzeit und benötigte Verwaltungsdaten (Overhead) mit sich, darüber hinaus erhöht es aber die Skalierbarkeit des Systems und reduziert die Komplexität.

- **Code on Demand (COD)**: Ist eine optionale Eigenschaft eines Restful Web-Service und bezeichnet die Bereitstellung von ausführbarem Softwarecode vom Server zum Client. Der Server übermittelt dem Client demnach ausführbaren Code, welcher nach Bedarf vom Client selbst ausgeführt wird, wie beispielsweise JavaScript oder Java-Applets.

Für den eigentlichen Austausch der Informationen setzten die meisten Web-Applikationen in der Vergangenheit auf das Simple Object Access Protocol, kurz SOAP (Hoffman, 2020). Dieses Protokoll forcierte die Verwendung von XML als Datenstruktur in einer strikt vorgegebenen Form, in welcher der tatsächliche Inhalt der Daten (Body) gemeinsam mit optionalen Kopfzeilen (Headers) in einem übergeordneten „Briefumschlag“ (Envelope) verpackt ist (Gudgin et al., 2007).

Die Grundstruktur einer solchen SOAP-Nachricht ist in Abbildung 2.2 dargestellt. Durch diese spezielle Struktur wurde das bereits „schwer“ interpretierbare XML Format zusätzlich verkompliziert.

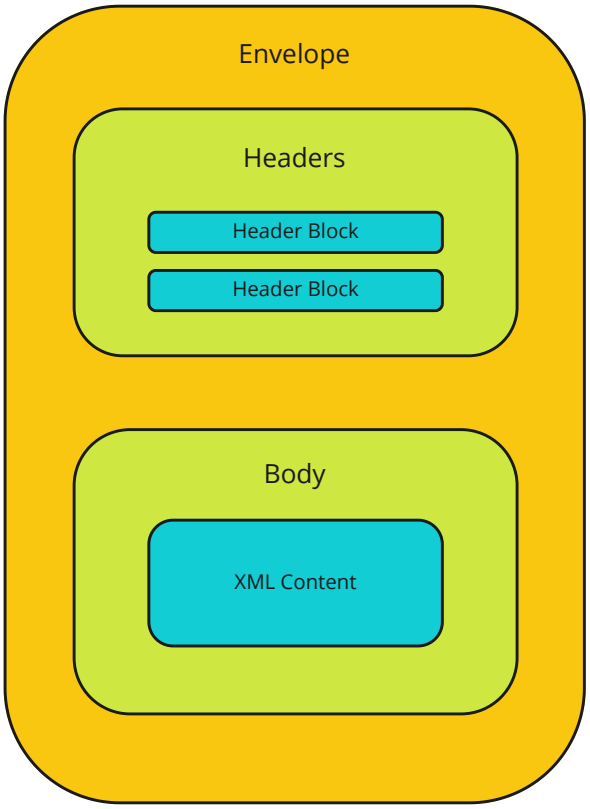


Abbildung 2.2: Struktur einer SOAP Nachricht[4]

Moderne Web-Applikationen sind heutzutage allerdings in der Lage durch die Verwendung von auf REST basierten Application Programming Interfaces (API) beliebige Formate für den Austausch von Informationen zu verwenden. In den meisten Fällen wird in der heutigen Zeit die JavaScript Object Notation (JSON) verwendet (Hoffman, 2020).

JSON ist im Vergleich zu XML wesentlich schlanker und weniger ausführlich aufgebaut als XML, wodurch die Interpretation des Informationsinhalts für den Menschen bedeutend vereinfacht wird (Hoffman, 2020). Listing 2.1 und 2.2 zeigen eine exemplarische Gegenüberstellung eines Objekts im XML und JSON Format:

[4] Grafik adaptiert von (IBM Coorporation, 2021).

```xml
<user>
  <name1>Bruce</name1>
  <name2>Wayne</name2>
  <dob>17.04.1915</dob>
  <addresses>
    <adr>
      <street1>Mountain Drive</street1>
      <street2>Wayne Manor</street2>
      <city>Gotham City</city>
      <country>DC</country>
      <zip>1007</zip>
    </adr>
  </addresses>
</user>
```

Listing 2.1: Beispiel XML Objekt

```json
{
  "name1": "Bruce",
  "name2": "Wayne",
  "dob": "17.04.1915",
  "addresses": [
    {
      "street1": "Mountain Drive",
      "street2": "Wayne Manor",
      "city": "Gotham City",
      "country": "DC",
      "zip": "1007"
    }
  ]
}
```

Listing 2.2: Beispiel JSON Objekt

Wie in Listing 2.2 zu sehen ist, besteht JSON grundsätzlich aus eindeutigen Schlüsseln (Keys) und zugeordneten Werten (Values). Über diese Key-Value Paare hinweg unterstützt es ebenfalls komplexe Datentypen wie beispielsweise Listen in einer leicht lesbaren Form. Für Maschinen sind jedoch sowohl XML als auch JSON Formate leicht zu interpretieren und daher kommt auch XML weiterhin noch in vielen REST basierten Applikationen zur Anwendung.

Neben der erleichterten Lesbarkeit von JSON bietet es allerdings noch weitere Vorteile im Vergleich zu anderen Formaten (Hoffman, 2020):

- Durch das entsprechend schlanke Format reduziert es die Netzwerk-Last.
- Es ist leicht zu parsen und reduziert dadurch die Hardwarelast der Server und Clients.
- Komplexe Datenstrukturen können hierarchisch dargestellt werden.
- Wird von allen gängigen modernen Browsern unterstützt.

Nachdem nun ein grober Überblick über die Funktionsweise moderner Web-Applikationen und deren Formate für den Informationsaustausch gegeben wurde, kann in weitere Folge ein Blick auf bekannte Attacken und Bedrohungen geworfen werden, vor welchen ein solches System in der heutigen Zeit geschützt werden muss.

Cross-Site Scripting

Eine der heutzutage am häufigsten verwendeten Attacken gegen Web-Applikationen ist Cross-Site Scripting. XSS basiert auf der zunehmenden Interaktion zwischen Web-

Anwendungen und deren Benutzerinnen und Benutzern und nutzt die Tatsache, dass innerhalb der verwendeten Browser Skripte durch die entsprechende Anwendung ausgeführt werden können. Solche Skripte können manipuliert werden und stellen demnach ein Risiko für die Anwendungen und deren Benutzerinnen und Benutzer dar (Hoffman, 2020).

Das Komitee des Open Web Application Security Project (OWASP) gliedert Cross-Site Scripting Attacken in drei unterschiedliche Kategorien (Hoffman, 2020):

- Persistent (stored)
- Nicht-Persistent (reflected)
- DOM-basiert (DOM-based)

Persistente (stored) XSS Angriffe treten in der Regel dann auf, wenn Eingaben von den Anwenderinnen und Anwendern auf dem Ziel-Server gespeichert werden müssen, beispielsweise in einer Datenbank, einem Forum oder Kommentarfeld. Die Benutzerinnen und Benutzer sind in weiterer Folge in der Lage, Daten aus der Web-Anwendung aufzurufen, ohne dass diese Daten zuvor für eine sichere Darstellung im Browser aufbereitet wurden. Der Server sendet diese manipulierten Daten in weiterer Folge an jeden Client, der diese anfordert (OWASP Foundation, 2022).

Nicht-Persistente (reflected) XSS Angriffe hingegen senden die Eingaben der Anwenderinnen und Anwender direkt an die entsprechende Web-Applikation, welche die Eingaben sofort auswertet und in einem Ergebnis oder einer Fehlermeldung zurückmeldet. Dadurch, dass die entsprechende Eingabe direkt vom Server zurückgemeldet wird, können schadhafte Skripte als Eingabe übergeben werden (OWASP Foundation, 2022). Sobald der Server die Antwort liefert, wird das entsprechende Skript vom Browser ausgeführt. Bringt ein solcher Angriff nun eine Anwenderin oder einen Anwender dazu, denselben URL mit der manipulierten Eingabe aufzurufen, wird es Angreifern ermöglicht, beliebiges JavaScript im Browser des entsprechenden Clients auszuführen und in weiterer Folge erheblichen Schaden zu verursachen (PortSwigger Ltd., o. J.).

Bei **DOM-basierten (DOM-based)** XSS Angriffen, auch lokales XSS genannt, findet der gesamte Datenfluss im Browser statt und erreicht in den meisten Fällen den Server nicht, wodurch in erster Linie statische Websites davon betroffen sind. Dabei wird darauf gesetzt, dass ein URL von der Benutzerin oder dem Benutzer aufgerufen wird. In diesem URL wird ein schadhaftes Skript platziert, beispielsweise als Abfrage-Parameter (Query Parameter) oder URL-Fragment. Sobald der entsprechende manipulierte Link

aufgerufen wird, wird das enthaltene Skript ohne Überprüfung direkt im Browser des Clients ausgeführt (IONOS SE, 2019).

Cross-Site Scripting kann demnach dazu verwendet werden, um mithilfe von Skripten Benutzereingaben mitzulesen, Daten zu manipulieren oder Zugriff auf gesamte Benutzer-Accounts zu erlangen. Im Jahre 2018 beispielsweise, war die zweitgrößte Fluggesellschaft Großbritanniens Opfer einer XSS-Attacke, bei welcher die Buchungsinformationen von rund 400.000 Transaktionen von Dritten aufgezeichnet wurden. Auch das bekannte Unternehmen eBay war bereits mehrmals von solchen XSS-Attacken betroffen. Im Jahre 2015 konnte durch Manipulation eines URL-Parameters Zugriff auf Produkte von Verkäufern erlangt werden und auch sensible Zahlungsinformationen konnten ausgelesen werden (Anderson, 2020).

SQL-Injection

Eine weitere bekannte Bedrohung für moderne Web-Applikationen nennt sich SQL-Injection. SQL steht für Structured Query Language und beschreibt die Sprach-Syntax, die für den Zugriff auf Datenbanken verwendet wird. Eine Datenbank kann man als großen Software-Container beschreiben, der eine große Menge an Daten in einer strukturierten Form speichern und zugänglich machen kann. Es gibt verschiedene Ansätze und Methoden um dies zu erreichen. Einer der gebräuchlichsten Ansätze ist das relationale Modell, bei welchem Daten in Form von Objekten mit Beziehungen untereinander dargestellt werden. Bekannte Ausführungen von relationalen Datenbanken sind beispielsweise Microsoft SQL, MySQL, SQLite und Oracle (Galluccio, Casell & Lombari, 2020).

Eine mögliche Alternative zu relationalen Datenbanken sind nicht-relationale Datenbanken. Solche Datenbanksysteme können auf den folgenden unterschiedlichen Ansätzen basieren (Galluccio et al., 2020):

- Netzwerk Datenbanken
- Graphenbasierte Datenbanken
- Objektorientierte Datenbanken
- Dokumentbasierte Datenbanken

Unabhängig vom verwendeten Modell haben all diese Systeme den Zweck, Daten zu speichern und zugänglich zu machen. Des Weiteren kann jedes dieser Datenbankmodel-

le von solchen SQL-Injection Angriffen betroffen sein, auch wenn die dahinter liegende Syntax nicht SQL entspricht (Galluccio et al., 2020). Im Kontext dieses Abschnittes wird der Fokus allerdings auf relationale Datenbanken und SQL gelegt.

SQL ist eine Sprache, die sehr leicht zu verwenden und zu verstehen ist und bietet darüber hinaus eine Vielzahl an Operationen für das Speichern, Lesen, Modifizieren und Löschen von Informationen. Die bekannteste Operation von SQL ist wohl das *SELECT*-Statement zum Lesen von Daten (Galluccio et al., 2020). Ein SQL SELECT-Statement ist wie folgt aufgebaut:

```
SELECT first_name, last_name, email
FROM user;

SELECT first_name, last_name, email
FROM user
WHERE id = 123;
```

Listing 2.3: Einfaches SQL Select Statement

Das erste SELECT-Statement in Listing 2.3 liest beispielsweise alle Einträge der Tabelle *user* und limitiert die Datensätze auf die Attribute Vorname (first_name), Nachname (last_name) und E-Mail (email). Das zweite Statement hingegen limitiert die Ausgabe darüber hinaus auf den Datensatz mit der ID *123*.

Neben der im zweiten Statement verwendeten Where-Klausel, können ebenfalls noch weitere verwendet werden wie *ORDER BY* zum Sortieren der Ausgabe auf Basis gewisser Attribute, *DISTINCT* zum Entfernen von Duplikaten oder *GROUP BY* um die Ergebnisse zu gruppieren (Galluccio et al., 2020).

Abgesehen vom SELECT-Statement für Lese-Zugriff, existieren weitere Befehle für die Erstellung, Änderung oder Löschung von Einträgen (Galluccio et al., 2020):

- INSERT/DELETE: Datensätze einer Tabelle hinzufügen/löschen
- CREATE/DROP: Zum Erstellen/Löschen ganzer Tabellen oder Datenbanken
- ALTER: Zum Ändern der Tabellen- oder Datenbank-Struktur.

SQL bietet demnach die Möglichkeit, auf verschiedenen Ebenen mit der gesamten Datenbank zu interagieren. Die Datenbank an sich ist die tatsächliche Quelle der Daten und der unbefugte Zugriff in Kombination mit SQL bietet potentiellen Angreifern uneinge-

schränkte Kontrolle über das Datenbank-System und ermöglicht es ihnen, das System oder die Urheber dieser Daten zu schädigen (Galluccio et al., 2020).

Aus diesem Grund sollte SQL-Code nie in direkter Interaktion mit einer Anwendung stehen. Stattdessen sollte die Anwendung nach einer Eingabe den SQL-Code vorbereiten und erst dann an die Datenbank senden, um die angeforderten Daten zu extrahieren oder zu ändern. Trotzdem eliminiert das nicht die Gefahr, dass die SQL-Syntax von potentiellen Angreifern missbraucht wird (Galluccio et al., 2020).

SQL-Injection fällt grundsätzlich in die Kategorie der Code-Injection und beinhaltet das Einfügen von Code, welcher von einer Anwendung verarbeitet wird und somit zur Ausführung nicht vorhergesehener Aufgaben führt (Galluccio et al., 2020).

Listing 2.4 zeigt eine Query, in welche mithilfe von SQL-Injection ein zerstörerischer SQL-Befehl eingeschleust wurde. Durch die Eingabe der ID *69* sollen in diesem Beispiel alle Daten dieses Benutzers angezeigt werden. In der Eingabe befindet sich allerdings neben der ID ebenfalls der Term *;DROP TABLE users;*, welcher nach Exekution die gesamte Benutzer-Tabelle löschen würde.

```
SELECT * FROM users WHERE id = 69; DROP TABLE users;
```

Listing 2.4: Beispiel SQL Injection

Werden Eingaben, die an eine Datenbank oder einen anderen Service gesendet werden nicht auf solchen Code geprüft, kann das unrechtmäßig gewährte Zugriffe bis hin zu völliger Zerstörung datenbankgestützter Webapplikationen zur Folge haben. Weitere Folgen von SQL-Injection sind (Galluccio et al., 2020):

- Offenlegen von Informationen der Datenbank oder ihres Inhalts
- Unautorisierte Änderungen an der Datenbank oder der Daten
- Privilegierter Zugriff auf Applikationen
- Einschränkung der Funktionalität von Anwendungen

Um dem entgegenzuwirken sollten Anwendungen bereits sicherheitsorientiert entwickelt werden und sowohl jegliche Textzeichenfolge vor der Auswertung prüfen, keine Sonderzeichen in Abfragen zulassen, nur bestimmte Abfragen erlauben und Freiheiten der Benutzerinnen und Benutzer einschränken (Galluccio et al., 2020).

Im Jahr 2007 beispielsweise, wurde die größte amerikanische Supermarktkette 7-Eleven Opfer einer SLQ-Injection-Attacke einer russischen Hacker-Gruppe. Die angreifende Gruppe erlangte mithilfe der Attacke Zugriff auf die Webseite und die dahinterliegende Datenbank des Unternehmens und konnte somit Zahlungsinformationen der Kundinnen und Kunden auslesen und zwei Millionen Dollar erbeuten (Malwarebytes Inc., o. J.).

Auch einer der größten Datendiebstähle unserer Zeit wurde mithilfe von SQL-Injection durchgeführt. Im Jahr 2017 wurde das amerikanische Unternehmen Equifax von einer solchen Attacke heimgesucht. Dabei wurden sensible Informationen wie Namen, Adressen, Sozialversicherungsnummern, Führerscheinnummern aber auch Kreditkarten-Informationen und andere sensible Dokumente von rund 150 Millionen Kunden offengelegt. Die Angreifenden nutzten dabei eine Sicherheitslücke in einem Framework der Web-Applikation des Unternehmens (Malwarebytes Inc., o. J.).

SQL-Injection setzt demnach am Kern der Informationen, der Datenbank, an und kann gravierende Folgen für die betroffenen Unternehmen und Personen haben. Anhand der scheinbar einfachen aber mächtigen Sprache SQL ist es demnach möglich, unautorisierten Zugriff auf Informationen oder auch ganze Systeme zu erhalten. Aus diesem Grund bedarf es besonderen Schutzvorkehrungen, um diesen Kern der Informationen vor unzulässigem Zugriff und dem Missbrauch durch Dritte zu schützen.

Cross Site Request Forgery (CSRF)

Eine weitere weit verbreitete kritische Schwachstelle für Web-Applikationen ist **Cross Site Request Forgery (CSRF)**. CSRF wird auch als „Session Riding“ oder XSRF bezeichnet und nutzt die Zustandslosigkeit von Web-Applikationen, um Benutzerinteraktionen auf einer Webseite zu simulieren. Es wird in der Regel dafür verwendet, um gewisse Aktionen, unter der Verwendung der authentifizierten Sitzung der Anwenderinnen und Anwender der Web-Applikation, auszuführen (Blatz, 2007).

Das häufigste Ziel von CSRF besteht darin, authentifizierte Aktionen im Namen anderer auszuführen, kann aber ebenfalls für eine Vielzahl anderer Angriffe verwendet werden. Beispielsweise kann eine CSRF-Attacke für die manipulative Teilnahme an Online-Umfragen verwendet werden oder sogar Distributed Denial Of Service (DDoS) Angriffe starten. Schlussendlich wird CSRF dazu verwendet, eine unsichere Vertrauensbeziehung zwischen Browser und Web-Server zu missbrauchen (Blatz, 2007).

Die Anzahl an Anfragen spielt bei CSRF keine Rolle und es lässt sich gut mit anderen Angriffen kombinieren. Beispielsweise kann CSRF in Kombination mit XSS oder SQL-

Injection verwendet werden. Dabei wird CSRF für die vermeintliche Authentifizierung verwendet und XSS oder SQL-Injection für den tatsächlichen Angriff (Blatz, 2007).

Ein CSRF Angriff kann auf unterschiedliche Arten erfolgen. Einerseits kann über sogenannte **Inline Links** ein Verweis auf eine externe Ressource, wie beispielsweise ein Bild, in eine andere Seite eingebettet werden. Dabei enthält das Bild die Information des Requests an den Ziel-Server und der Browser kann nicht erkennen, dass es sich bei dem angeforderten Bild um eine Webseite handelt (Blatz, 2007).

Eine weitere Art eines CSRF-Angriffs kann über **automatisch übermittelte Web-Formulare (Auto-Submitted Forms)** erfolgen. Dazu muss die angreifende Person Kontrolle über die Webseite haben oder sich über eine XSS-Schwachstelle Zugriff verschaffen, um manipuliertes HTML einbetten zu können. Grundsätzlich kann jede Webseite mit einer XSS-Schwachstelle auch für CSRF-Angriffe genutzt werden (Blatz, 2007).

Eine weitere bekannte Art von CSRF-Attacken ist **Phishing**. Beim Phishing hat die angreifende Person die vollständige Kontrolle über die Webseite und zwingt seine Opfer die Seite zu besuchen. Die Besucherinnen und Besucher der Webseite werden dazu gebracht Informationen preiszugeben und die angreifende Partei kann diese Informationen durch Aktionen auf der Ziel-Webseite ergänzen. Auch wenn sich die Benutzerinnen und Benutzer der Webseite weigern Informationen preiszugeben, könnten diese bereits beim Aufruf der Seite gefährdet sein. Phishing ist gerade in Verwaltungs- und Intranet-Systemen sehr effektiv. Beispielsweise können E-Mails versendet werden, die vorgeben, aus dem eigenen oder einem Partnerunternehmen zu kommen und dadurch die authentifizierte Session beim Aufrufen einer Website nutzen, um weitere Aktionen auszuführen (Blatz, 2007).

Mithilfe dieser unterschiedlichen CSRF-Attacken können demnach eine Vielzahl an Aktionen im Namen der eingeloggten Person erfolgen. Einige Beispiele für Aktionen sind folgende (Blatz, 2007):

- Erstellung/Übernahme von Accounts
- Genehmigung von Transaktionen wie Überweisungen oder Aktienkäufe
- XSS und SQL-Injection Angriffe
- Aufrufen anderer Web-Services

Um sich vor CSRF-Angriffen zu schützen, können Systeme Gebrauch von *Nonces* machen, wie in Kapitel 2.1.1 bereits beschrieben. Zum einen können diese Nonce dabei in Web-Fomulare eingebettet und serverseitig überprüft werden, sobald das Formular übermittel wurde (Single-Per-Page-Nonce). Zum anderen können sogenannte Per-Session-Nonce verwendet werden. Diese Nonce werden beim Aufbau der Browser-Session erstellt und über die gesamte Session hinweg verwendet. Der Server vergleicht diese Nonce bei jedem Request und kann so unautorisierte Anfragen blockieren. Eine weitere beliebte Schutzvorkehrung gegenüber CSRF-Angriffen sind Bestätigungsdialoge oder CAPTCHAs. CAPTCHAs verhindern das Versenden von Web-Formularen durch automatisierte Skripte, indem die eingeloggte Person eine Aktion tätigen muss, die von Maschinen nur schwer ausgeführt werden kann. Beispiele dafür sind das Auswählen bestimmter Bilder oder die Eingabe von verschwommenen Texten (Blatz, 2007).

Allerdings können auch die Benutzerinnen und Benutzer selbst zum Schutz vor CSRF beitragen. Durch das explizite Ausloggen von Web-Anwendungen, Änderung von Passwörtern, Verwendung unterschiedlicher Browser oder der Verwendung von virtuellen Maschinen oder Proxies, werden CSRF-Angriffe für die angreifende Partei bedeutend erschwert (Blatz, 2007).

Auch von CSRF waren bereits bekannte Unternehmen betroffen. Im Jahr 2010 war Twitter und zehntausende Benutzerinnen und Benutzer Opfer einer vergleichsweise harmlosen CSRF-Attacke. Aufgrund fehlender Maßnahmen gegen CSRF wurden die Anwenderinnen und Anwender auf eine Seite gelockt, welche anhand von CSRF Tweets im Namen der registrierten Person erstellte und darin weitere Benutzer auf dieselbe Seite geführt hat. Neben Twitter waren ebenfalls Größen wie Facebook, eBay oder PayPal von CSRF-Schwachstellen betroffen (Ferguson, 2017).

2.2 Konzepte und Patterns

Im Laufe der Zeit haben sich gewisse Konzepte und Patterns etabliert, um die Sicherheit von (Web-) Anwendungen zu erhöhen. Diese bilden ein wichtiges Instrument, um die Entstehung potentieller Sicherheitslücken abzuschwächen oder zu vermeiden (Kohnfelder, 2021).

Laut Kohnfelder (2021) lassen sich solche Security-Patterns in folgende Gruppen unterteilen:

- Design Eigenschaften (Design Attributes)
- Minimale Offenlegung (Exposure Minimization)
- Vertrauen & Verantwortung (Trust & Responsibility)
- Redundanz (Redundancy)
- Starke Durchsetzung (Strong Enforcement)

In den folgenden Abschnitten soll demnach ein Einblick in die einzelnen Gruppen gegeben werden, um deren Konzepte in Bezug auf Sicherheit von (Web-) Applikationen zu erläutern.

2.2.1 Design Eigenschaften

Die **Design Eigenschaften (Design Attributes)** sollen aufzeigen, wie ein gutes und sicheres Design aussehen soll: einfach und transparent. **Ökonomisches Design (Economy of Design)** soll die Sicherheitsanforderungen dahingehend erhöhen, dass weniger Fehler und unentdeckte Schwachstellen vorhanden sind. Beispielsweise wird in der Systemarchitektur von großen Web-Anwendungen die Funktionalität in kleine, in sich geschlossene Komponenten zerlegt, die gemeinsam komplexe Vorgänge ausführen und so die Zuverlässigkeit der Anwendung gewährleisten und erhöhen (Kohnfelder, 2021).

Dieses Prinzip besagt jedoch nicht, dass der einfache Weg auch immer der bessere ist. Es erhöht allerdings die Wartbarkeit der Anwendung durch die Entwickler und verringert den Lernaufwand für die Benutzerinnen und Benutzer. Als Beispiel nennt Kohnfelder (2021) Access Control Lists (ACL) unter Unix und Windows. Unter Unix werden die Zugriffsrechte für Dateien auf Benutzer oder Gruppenebene für alle festgelegt. Windows hingegen setzt auf eine beliebige Anzahl an Einträgen zur Zugriffskontrolle, welche darüber hinaus vererbt und priorisiert werden können (Kohnfelder, 2021).

Transparentes Design (Transparent Design) beruht darauf, dass bei der Enthüllung des Designs die Unbesiegbarkeit des Systems aufgezeigt wird und die Angreiferinnen und Angreifer davon abgeschreckt werden. Das heißt allerdings nicht, dass die System-Entwürfe offengelegt werden müssen. Wenn durch Offenlegung des Designs die Schwachstellen des Systems enthüllt werden, sollte das Design angepasst und nicht einfach geheim gehalten werden (Kohnfelder, 2021).

Ein gutes Beispiel hierfür ist die Zerstörung des Todessterns aus Star Wars. Die offenlegung des schlechten Designs ermöglichte es den Rebellen, die Schwachstelle des Sterns ausfindig zu machen und ihn dadurch in der Schlacht von Yavin zu zerstören (Kohnfelder, 2021)

2.2.2 Minimale Offenlegung

Das Prinzip der **minimalen Offenlegung (Exposure Minimization)** zielt darauf ab, auf Nummer Sicher zu gehen. Bei der Ausführung von Aufgaben soll darauf geachtet werden, immer die **geringsten Privilegien (Least Privilege)** zu verwenden, um das Risiko von Fehlern zu minimieren. System-Administratoren haben in der Regel mächtige Zugriffsrechte und sollten demnach nicht wahllos im Internet surfen. Genau so sollte man eine geladene Waffe nicht reinigen, ohne sie davor zu entladen. Angreifende werden so beim Ausnutzen einer Sicherheitslücke in ihren Möglichkeiten eingeschränkt. Mächtige Zugriffsrechte sollten nur dort eingesetzt werden, wo es wirklich notwendig ist und nur für die kürzest mögliche Dauer. Zwar kostet die selektive Zuweisung der Zugriffsrechte Zeit, kann jedoch im Falle eines Exploits bedeutende Wirkung haben. Es macht einen großen Unterschied, ob ein geringfügiger Eingriff in ein System erfolgt, oder das gesamte System kompromittiert wird (Kohnfelder, 2021).

Analog zur Verwendung der geringsten Privilegien ist es ebenfalls erstrebenswert, nur so viele Informationen zu verwenden, wie für die Abwicklung der Aufgabe auch wirklich notwendig sind **(Least-Information)**. Dadurch kann verhindert werden, dass Risiken durch unbeabsichtigt offengelegte Informationen entstehen. Wird dieses Pattern nicht schon beim Design der Software berücksichtigt, kann es später extrem schwierig zu implementieren sein, da jeweils beide Seiten der Schnittstellen angepasst werden müssen (Kohnfelder, 2021).

In modernen Web-Applikationen kann beispielsweise GraphQL für den Austausch von Informationen nach dem Least-Information Pattern verwendet werden. GraphQL ist eine Abfrage-Sprache (Query Language) für APIs und basiert typischerweise auf REST und dem HTTP-Protokoll. Über eine solche GraphQL-Query können genau die Informationen abgefragt werden, die tatsächlich benötigt werden (Porcello & Banks, 2018).

Listing **??** und 2.6 zeigen eine einfache GraphQL-Anfrage und deren Antwort. In der Anfrage können die benötigten Attribute des Datensatzes definiert werden. Hier etwa der Name, der Name der Heimatwelt (homeworld) und der Titel der Filme (filmConnection) der Person mit der personID vier.

```
query{
    person(personID: 4){
        name
        homeworld{
            name
        }
        filmConnection{
            edges{
                node{
                    title
                }
            }
        }
    }
}
```

Listing 2.5: GraphQL-Anfrage [5]

[5]Beispieldaten von (The GraphQL Foundation, o. J.).

```
{
  "data": {
    "person": {
      "name": "Darth Vader",
      "homeworld": {
        "name": "Tatooine"
      },
      "filmConnection": {
        "edges": [
          {
            "node": {
              "title": "A New Hope"
            }
          },
          {
            "node": {
              "title": "The Empire Strikes Back"
            }
          },
          {
            "node": {
              "title": "Return of the Jedi"
            }
          },
          {
            "node": {
              "title": "Revenge of the Sith"
            }
          }
        ]
      }
    }
  }
}
```

Listing 2.6: GraphQL Antwort [6]

Eine Software sollte prinzipiell so entworfen werden, dass sie standardmäßig sicher ist **(Secure by Default)**, sodass Untätigkeiten von Benutzerinnen und Benutzern kein Risiko darstellen. Ein unsicherer Zustand sollte niemals mit der Absicht implementiert werden, ihn später abzusichern. Dadurch entsteht ein Intervall der Verwundbarkeit und es kann leicht vergessen werden. Dieser Pattern gilt für jegliche Einstellung oder

[6] Beispieldaten von (The GraphQL Foundation, o. J.).

Konfiguration, die sich nachteilig auf die Sicherheit auswirken kann. Jedenfalls sollte man davon absehen Standardpasswörter zu verwenden und darüber hinaus Berechtigungen auf restriktive Einstellungen setzen, potentiell gefährliche Optionen deaktivieren und Schutz-bietende Sicherheitsoptionen aktivieren (Kohnfelder, 2021).

Ein weiteres Pattern angelehnt an das Prinzip der minimalen Offenlegung ist die Verwendung von sogenannten **Allow-Lists gegenüber Block-Lists (Allowlists over Blocklists)**. Eine Allow-List ist eine endliche Menge dessen, was sicher ist. Eine Block-List hingegen ist eine Aufzählung dessen, was unsicher ist. Diese erlaubt implizit eine unendliche Menge an *Dingen*, die unsicher sind. Das Problem bei diesen Block-Lists ist, dass jeder Eintrag der ausgelassen wird, zu einem Fehler wird. Die Verwendung einer unvollständigen Allow-List hingegen öffnet keine Sicherheitslücken, die riskante Aktivitäten ermöglicht (Kohnfelder, 2021).

Die **Berechenbarkeit (Avoid Predictability)** von Systemen lehnt sich an das gleiche Prinzip an. Jegliche Daten oder Verhaltensweisen, die vorhersehbar sind, können nicht geheim gehalten werden, sondern von Dritten erraten werden. Dies kann schwerwiegende Folgen haben, da dadurch wichtige oder sensible Informationen preisgegeben werden können. Beispielsweise wird bei der Registrierung auf einer Website eine ID für das Kundenkonto vergeben. Die Einfachste Möglichkeit dies zu tun ist, das erste Konto mit der ID eins zu kennzeichnen. Das zweite Konto bekommt die ID zwei und so weiter. Dies würde beispielsweise Rückschlüsse auf die Anzahl der registrierten Konten liefern. Etwas, das gegenüber der Konkurrenz gerne geheim gehalten wird. Wird so eine ID beispielsweise aus den Stammdaten des entsprechenden Kontos generiert, würde dies wiederum sensible Informationen der Inhaberinnen und Inhaber des Kontos preisgeben. Für solche grundlegenden Daten sollten demnach eher Zufallszahlen, gegebenenfalls in Kombination mit zufälligen Buchstaben, verwendet werden (Kohnfelder, 2021).

Das letzte Pattern der minimalen Offenlegung, das in dieser Arbeit behandelt wird, ist das **sichere Scheitern (Fail Securely)**. Bei Auftritt eines Problems sollte sich das System in einem sicheren Zustand befinden und bestenfalls fortgesetzt werden können oder geordnet heruntergefahren werden. Angreifende neigen dazu, solche Fehlerfälle absichtlich auszulösen und so Schwachstellen im System aufzudecken, die sie in weiterer Folge ausnutzen können. Daher sollte jeder Fehler sicher behandelt werden oder zur vollständigen Ablehnung der Anfrage führen (Kohnfelder, 2021).

2.2.3 Starke Durchsetzung

Durch **starke Durchsetzung** soll sichergestellt werden, dass die Abläufe im Quell-Code (Source-Code) sich an die vorgegebenen Regeln der Anforderung halten. Es soll vermieden werden, Möglichkeiten zu schaffen, mit denen das System umgangen werden kann und es unmöglich machen, verbotene Operationen auszuführen (Kohnfelder, 2021).

Dabei sollen durch **vollständige Mediation (Complete Mediation)** alle Zugriffe auf Daten einheitlich und sicher geprüft werden. Auch wenn es mehrere Zugriffsmöglichkeiten gibt, müssen all diese über die gleiche Form der Berechtigungsprüfung verfügen. Um dem gerecht zu werden, sollte es beim Zugriff auf Informationen einen einzigen Zugriffspunkt geben, an dem eine Sicherheitsentscheidung getroffen werden muss. Ist dies nicht möglich, sollten alle Zugänge an denen der Datenzugriff erfolgt, über eine funktional gleichwertige Sicherheitsüberprüfung verfügen. In der praktischen Umsetzung dieses Patterns gibt es unterschiedliche Grade der Konformität in Bezug auf die starke Durchsetzung. Wenn der Zugriff auf die Ressourcen nur über eine gemeinsame Routine erlaubt ist, spricht man von hoher Konformität. Bei Zugriffen über verschiedene Stellen, die jedoch alle über funktional gleichwertige Überprüfungen verfügen, ist mittlere Konformität gegeben. Bei inkonsistenten Sicherheitsüberprüfungen über verschiedene Stellen spricht man von geringer Konformität (Kohnfelder, 2021).

Ein weiteres Pattern, dass sich auf das Prinzip der starken Durchsetzung stützt, ist der **geringste gemeinsame Mechanismus (Least Common Mechanism)**. Dabei sollen unabhängige Prozesse voneinander isoliert werden, indem die Anzahl der gemeinsamen Mechanismen reduziert wird. Um dies zu gewährleisten, sollten Dienste entwickelt werden, die als Schnittstelle zu den unabhängigen Prozessen oder Benutzerinnen und Benutzern fungieren. Beispielsweise kann anstatt einer Datenbank für alle, eine kontextabhängige Datenbank pro Benutzerin und Benutzer bereitgestellt werden. Ein modernes Beispiel für das Prinzip des geringsten gemeinsamen Mechanismus sind Web-Cookies. Dabei speichert jeder Client seine eigenen Daten unabhängig von anderen Clients in Cookies (Kohnfelder, 2021).

2.2.4 Redundanz

Redundanz bildet eine Kernstrategie für die Sicherheit in Systemen und kann durch Verteidigung in der Tiefe (Defense in Depth) oder durch Abgrenzung von Privilegien

(Separation of Privilege) erreicht werden (Kohnfelder, 2021).

Verteidigung in der Tiefe (Defense in Depth) versucht durch Kombination unabhängiger Schutzschichten, den Gesamtschutz eines Systems zu erhöhen und so kritische Sicherheitsentscheidungen durchzusetzen. Dies erfordert zwar erhöhten Aufwand in der Implementierung und kann die Laufzeit erhöhen, jedoch sinkt dadurch die Wahrscheinlichkeit, dass ein Angriff jede Schutzschicht durchdringt und Schaden verursachen kann. Als Beispiel kann eine Sandbox genannt werden, die zur Ausführung von unsicherem Code verwendet wird. Dieser Code wird vor der Ausführung in der Sandbox von einer anderen Schicht nach gewissen Regeln validiert. Nur bei positiver Validierung wird der Code auch tatsächlich ausgeführt. Scheitert die Validierung, wird der Code vom System zurückgewiesen (Kohnfelder, 2021).

Abgrenzung von Privilegien (Separation of Privilege) ist auch bekannt als *Trennung der Aufgaben* und bezieht sich auf die Tatsache, dass zwei Schlösser, die unterschiedliche Schlüssel haben, die wiederum von unterschiedlichen Parteien gehalten werden, stärker sind, als ein Schloss alleine. Dabei soll prinzipiell Verantwortung in mehrere Aufgaben unterteilt werden, um so Schäden durch Fehler oder böswilligen Absichten zu vermeiden. Beispielsweise kann eine Sicherungskopie von Daten mit zwei verschiedenen Sicherheits-Schlüsseln verschlüsselt werden. Der Zugriff auf die tatsächlichen Daten ist dann nur möglich, wenn man im Besitz beider Schlüssel ist (Kohnfelder, 2021).

2.2.5 Vertrauen & Verantwortung

Moderne Software-Systeme sind in zunehmendem Maße miteinander verbunden und voneinander abhängig. Vertrauen und Verantwortung lassen diese Zusammenarbeit funktionieren.

Das Pattern **Zurückhaltung beim Vertrauen (Reluctance to Trust)** beschreibt Vertrauen als eine explizite, auf Beweisen beruhende Entscheidung. Ein gewisses Maß an Misstrauen sollte immer vorliegen, auch wenn keine Böswilligkeit besteht. Beispielsweise setzen Web-Server Cookies am Client und erwarten, dass diese mit der nächsten Anfrage erneut übermittelt werden. Da die Cookies aber gelöscht oder geblockt werden können, muss der Web-Server davon ausgehen, dass die Clients dieser Aufgabe nicht nachkommen (Kohnfelder, 2021).

Das Pattern **Verantwortung für die Sicherheit übernehmen (Accept Security Responsibility)** setzt hingegen bereits auf Vertrauen während der Entwicklung. Software

wird von Menschen entwickelt und die Menschheit ist von Natur aus vertrauenswürdig. Die Menschen, die die Software entwickeln, haben demnach die Pflicht, Verantwortung für die Sicherheit des Systems zu übernehmen. Bei zwei Systemen, die miteinander kommunizieren, muss demnach klar sein, welche Partei welche Verantwortung übernimmt und diese auch einhalten. Dies sollte über explizite Vereinbarungen geregelt werden. Eine Partei übernimmt beispielsweise die Eingabe-Validierung der Daten, wodurch sich die zweite Partei nicht mehr darum kümmern muss. Maximale Sicherheit kann allerdings über das bereits beschriebene Defense in Depth Pattern erreicht werden, indem beide Parteien die Validierung übernehmen (Kohnfelder, 2021).

2.3 Auslagerungsmöglichkeiten

Dadurch, dass Unternehmen bei der Abwicklung ihrer Geschäfte immer stärker auf das Internet angewiesen sind und gesetzliche Regelungen immer strenger werden, entwickelt sich das Management von Informationssicherheit immer weiter zu einer kritischen Unternehmensfunktion. Simultan erhöht sich die Komplexität der Informationssicherheit durch sich stetig verändernde Angriffsmuster, komplexe Infrastrukturen, Mangel an kompetentem Personal und begrenztem Budget. Das macht die Verwaltung der Informationssicherheit für viele Unternehmen zu einer wichtigen aber anspruchsvollen Aufgabe. Einige behelfen sich damit, sicherheitsrelevante Funktionen an externe Unternehmen auszulagern (A. Cezar, Cavusoglu & Raghunathan, 2010).

2.3.1 Kriterien

Zwar stehen einige Sicherheitsexperten der Auslagerung (Outsourcing) relevanter IT-Funktionen eher skeptisch gegenüber, andere hingegen sehen gute Gründe darin und befürworten das Outsourcing von traditionellen IT-Funktionen. Die Entscheidung, ob gewisse IT-Funktionen ausgelagert werden sollten, hängt dabei von einigen Faktoren ab, wie beispielsweise der Art der Funktion, den Kosten, der Qualität und dem Wettbewerbsumfeld. Wie es so oft beim Outsourcing der Fall ist, muss die Entscheidung dafür jedenfalls einen qualitativen und/oder kostenspezifischen Vorteil für das Unternehmen bringen (A. Cezar et al., 2010).

Einige Gründe, die das Outsourcing gewisser IT-Funktionen befürworten, sind beispielsweise (Gonzalez, Gasco & Llopis, 2010):

- Fokussierung auf die eigenen Kernkompetenzen.
- Erhöhung der Flexibilität und technologisch am aktuellsten Stand bleiben.
- Erhöhung der Qualität durch hochwertige externe IT-Dienstleistungen.
- Eliminieren von Routineaufgaben.
- Erleichterter Zugang zu (neuen) Technologien.
- Reduktion des Risikos, veraltete Technologien zu verwenden.
- Reduktion von Personal- und Technologiekosten.

Skeptiker, hingegen, berufen sich auf folgende Ausschluss-Kriterien, die das Outsourcing von IT-Funktionen zurückweisen (Gonzalez et al., 2010):

- Mangelhafte Qualifizierung des Provider-Personals.
- Nichteinhaltung von Verträgen oder Kundenbedürfnissen durch Auftragnehmer.
- Abhängigkeit zu externen Unternehmen.
- Verlust von technischem Know-How.
- Versteckte Kosten.
- Unklares Kosten-Nutzen-Verhältnis.
- Unumkehrbarkeit.

Entscheidet man sich dennoch für das Outsourcing von (sicherheits-) relevanten IT-Funktionen, kann man sich an diverse Serviceanbieter wenden.

2.3.2 Managed Security Service Provider

Managed Security Service Provider (MSSP) verwalten eine breite Palette von sicherheitsrelevanten Anforderungen sowohl für Klein- und Mittelunternehmen, als auch große Organisationen. Zu ihren Anwendungsgebieten zählen unter anderem die Verwaltung von Firewalls, Intrusion Detection Systemen (IDS)[7], virtuelle private Netzwerke (VPN),

[7] Systeme zur Überwachung und Analyse von Datenströmen (Grunwald & Herz, 2017).

Sicherheitsüberwachung, Incident Management[8], forensischen Analysen, Schwachstellenbewertung, Virenschutz und das Filtern von Inhalten (Hirschheim, Heinzl & Dibbern, 2020).

Managed Security Service Provider fungieren hierbei als helfende Hand für IT-Fachleute und nehmen teils sogar die Position eines virtuellen Chief Information Security Officer (vCISO) ein, um mögliche Risiken soweit als möglich zu reduzieren und zu vermeiden. Mittlerweile hat sich eine Vielzahl an Unternehmen am Markt etabliert, welche solche Managed Security Services anbieten. Einige Bekannte Unternehmen sind beispielsweise die Herjavec Group, Accenture, IBM oder Cisco (Cyber Defense Media Group, 2021).

2.3.3 Managed Detection and Response Provider

Während Managed Security Service Provider in vielen Fällen (vertragsabhängig) nur die Infrastruktur für die Reduktion von Sicherheitsrisiken bereitstellt und die eigentliche Problembehandlung den Kundeninnen und Kunden überlässt, bieten Managed Detection and Response (MDR) Provider genau die Reaktion auf eingehende Sicherheitsrisiken. Solche Provider konzentrieren sich proaktiv auf die Suche nach Bedrohungen, darüber hinaus auf deren Erkennung und eine angemessene Reaktion darauf (Kelley, 2021).

Sie verwalten in der Regel Tools zur Erkennung und Verwaltung von Risiken, wie beispielsweise Software zum Schutz von Endpunkten (Endpoint Protection Agents). Im Gegensatz zu MSSPs, welche stark auf automatisierte Dienste setzen, werden MDR Services für gewöhnlich aktiv von Menschen betrieben. Dabei werden die Netzwerke der Klientinnen und Klienten in Echtzeit auf Indikatoren für Angriffe (Indicators of Attack (IOA)) und Kompromittierungen (Indicators of Compromise (IOC)) überwacht, um Risiken frühzeitig zu erkennen und entsprechend reagieren zu können (Kelley, 2021).

In welchem Ausmaß ein MDR Provider auf Risiken reagiert, muss dabei allerdings klar vertraglich geregelt sein und ist neben der vertraglichen Basis ebenfalls von der Infrastruktur der Dienstnehmerinnen und Dienstnehmer abhängig. Die Reaktionsfähigkeit kann dabei vermeintlich einfache Aufgaben beinhalten, wie beispielsweise die Erkennung und automatisierte Entfernung von Malware, oder aber auch komplexe Aufgaben wie das Abschalten von Firewall-Ports oder virtuellen Maschinen. Die tatsächliche Reaktion des Providers kann, je nach Vertrag, von einfachen Warnungen über gezielte Aktionen bis hin zu einem umfassenden Prozess zur Eindämmung der Risiken reichen.

[8] Bereich des IT-Servicemanagement, um rasch auf Störungen zu reagieren (Gillis, 2018).

Der MDR Provider hat demnach in den meisten Fällen tiefen Einblick in die Ressourcen und (teils sensible) Daten des Unternehmens (Kelley, 2021).

Ebenso wie bei Managed Security Services gibt es ebenfalls im Bereich der Managed Detection and Response Services eine Vielzahl an etablierten Unternehmen am Markt. Unter anderem zählen Arctic Wolf, Red Canary, SecureWorks und FireEye zu solchen Dienstleistungsunternehmen (Braue, 2021).

3 Web Application Firewalls

Das Konzept, sich mithilfe einer Mauer vor Eindringlingen und Angriffen zu schützen, reicht bereits viele tausende Jahre in die menschliche Evolution zurück. Der Bau der Chinesischen Mauer, beispielsweise, sollte das chinesische Volk vor Angriffen der benachbarten Stämme schützen. Im Mittelalter errichteten die Machthaberinnen und Machthaber Mauern um ihre Schlösser, um sich selbst und das Volk in widerstandsfähigen Festungen vor Angriffen feindlicher Königreiche zu schützen. Der Begriff „Brandmauer“ (Firewall) trat etwas weiter in der Moderne auf und bezeichnete Mauern, welche Teile eines Gebäudes schützen sollten, in welchen die Wahrscheinlichkeit für das Entstehen von Bränden am höchsten war, wie etwa in Küchen. Mithilfe solcher Brandmauern konnte die Ausbreitung eines Feuers verlangsamt oder sogar verhindert werden (Ingham, Consulting & Forrest, 2002).

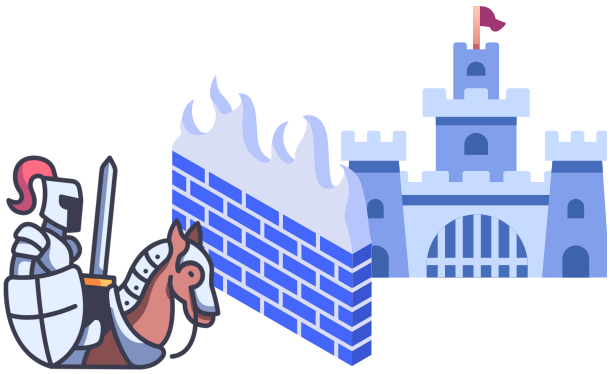


Abbildung 3.1: Der Ritter der Firewall[1]

Im modernen Umfeld des zwanzigsten und einundzwanzigsten Jahrhunderts bezieht sich der Begriff *Firewall* allerdings auf die Sicherheit in Computernetzwerken. Vorreiter für Firewalls waren in den späten 80er Jahren Netzwerk-Router, welche unterschiedliche Netzwerke voneinander trennten. Ein Problem im ersten Netzwerk konnte somit isoliert werden und das zweite Netzwerk weitgehend davor geschützt werden. Durch diese Beispiele wird deutlich, dass der Begriff Firewall verwendet wird, um einzelne Geräte oder Geräte-Gruppen vor potenziell gefährlichen externen Einflüssen zu schützen und die Ausbreitung von gefährlichen Ereignissen zu unterbinden oder zu verlangsamen (Ingham et al., 2002).

[1] Grafik vom Autor erstellt.

In diesem Kontext definiert sich eine Firewall als Grenze zwischen zwei Netzwerken, die den gesamten Verkehr zwischen diesen Netzwerken regelt und über einen Mechanismus verfügt, um Richtlinien durchzusetzen und gewisse Daten zu filtern oder zu blockieren. Dieses Vorgehen ist exemplarisch in Abbildung 3.2 dargestellt. Darüber hinaus sollte eine Firewall folgende Kriterien erfüllen (Ingham et al., 2002):

- Widerstand gegen sicherheitsrelevante Verstöße
- Ermöglichung von Audits und Monitoring von Ressourcen
- Kein direkter Benutzerzugang
- Starke Authentifizierung
- Ausfallsicherheit

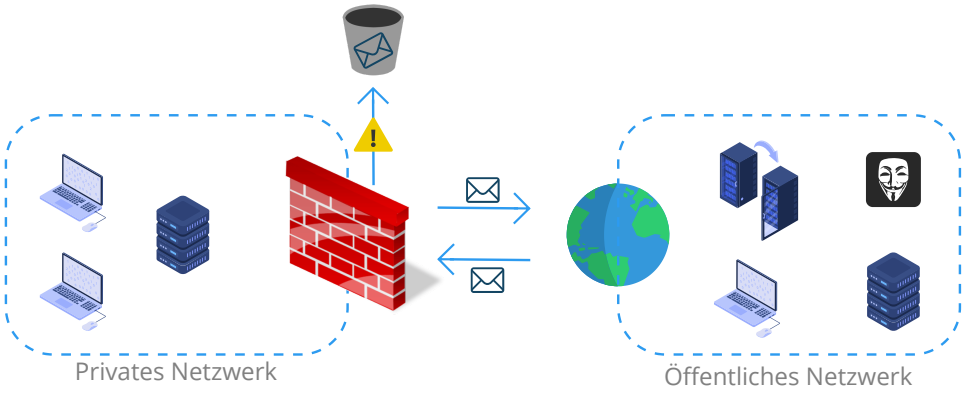


Abbildung 3.2: Netzwerk-Firewall[2]

Diese Grenze, die eine Firewall dabei einnimmt, kann dabei sowohl physisch, also ein Stück Hardware, als auch ein virtuelles Stück Software sein, die auf Client-Computern oder Servern installiert ist. Sie wird demnach dazu verwendet, um den Netzwerkverkehr auf schädliche Angriffe, Eindringlinge, Daten, Nachrichten oder andere Ereignisse für einzelne Clients oder ganze Netzwerke zu filtern. Meistens befinden sich Firewalls direkt am Rand eines Netzwerkes und schützen sowohl private Netzwerke als auch das Internet vor solchen Bedrohungen (Stewart & Kinsey, 2020).

Eine Firewall fungiert auf dem *Deny By Default / Allow By Exception* Prinzip und regelt nach definierten Richtlinien den ein- als auch ausgehenden Verkehr eines Netzwerkes. Dieses Prinzip besagt, dass keine Datenpakete an der Firewall vorbeikommen, ohne davor eine Reihe an Anforderungen zu erfüllen. Diese Anforderungen werden dabei

[2]Grafik vom Autor erstellt.

von den jeweilig zuständigen Netzwerkadministratoren oder Sicherheitsbeauftragten definiert. Da nicht jeder Datenverkehr von einer autorisierten Quelle stammt, muss dafür gesorgt werden, dass eingehender Traffic (Ingress) entsprechend gefiltert wird. Einhergehend kann auch nicht davon ausgegangen werden, dass der gesamte interne Datenverkehr für die Außenwelt bestimmt ist. Daher ist es wichtig ebenso für den ausgehenden Traffic (Egress) entsprechende Richtlinien aufzustellen und diesen bei Nichteinhaltung entsprechend zu blockieren (Stewart & Kinsey, 2020).

Für die tatsächliche Filterung des Datenverkehrs liest eine Firewall die einzelnen Datenpakete aus und untersucht die Kopfzeilen (Header), den Inhalt (Payload), das Aufbauen einer Verbindung (Session) oder auch andere Werte, wobei die meisten Firewalls sich auf eines dieser fokussieren. Im Header stehen dabei Informationen über die Quelle, den Typ, die Größe also auch die Absender- und Zieladresse des Datenpakets. In der Payload befinden sich die tatsächlichen Informationen. Erfüllt ein Datenpaket ein oder mehrere Richtlinien, wird der Zugriff von der Firewall autorisiert. Wird keine Richtlinie erfüllt, wird der Zugriff automatisch verweigert. Im Falle eines Ausfalls fungiert die Firewall nach dem Fail-Safe Prinzip und es wird automatisch der gesamte Traffic blockiert (Stewart & Kinsey, 2020).

Allerdings stoßen auch Netzwerkfirewalls an ihre Grenzen und können einige Risiken nicht berücksichtigen, wodurch sich Sicherheitslücken für das gesamte Netzwerk ergeben. Beispielsweise sind Firewalls keine Authentifizierungs-Systeme und demnach nicht dafür entworfen, um Zugangsdaten zu kontrollieren oder digitale Zertifikate zu verifizieren. Weiters sind Firewalls nicht in der Lage, verschlüsselten Datenverkehr einzusehen und dessen Inhalt auszulesen. Darüber hinaus können Firewalls keinen schadhaften Code innerhalb der übermittelten Daten erkennen. Firewalls agieren auf Basis definierter Regeln. Diese Regeln halten sich meist in Grenzen und limitieren sich auf einige Dutzend bis hundert Stück. Wären Firewall Regeln ebenfalls auf das Erkennen von schadhaftem Code ausgelegt, würde sich die Anzahl an definierten Regeln im Bereich von Millionen bewegen und stetig wachsen (Stewart & Kinsey, 2020).

Folglich können Firewalls ebenso wenig die nachfolgenden Funktionen übernehmen (Stewart & Kinsey, 2020):

- Sind keine Intrusion Detection Systeme
- Schützen nicht vor Insider-Attacken[3]

[3] Böswillige Absichten von internen Personen oder externen Unternehmen, die Zugriff auf die eigene Infrastruktur haben (Vodopyan, 2022).

- Schützen nicht vor Wechselmedien wie USB-Sticks
- Schützen nicht vor physischen Attacken
- Können falscher Konfiguration nicht vorbeugen

Einigen dieser Grenzen kann entgegengewirkt werden, indem auf sogenannte *Next Generation Firewalls (NGFW)* zurückgegriffen wird. Diese NGFWs erweitern das Konzept regulärer Netzwerkfirewalls um Funktionen von Intrusion Detection- und -Prevention Systemen und sind beispielsweise in der Lage, mit Authentifizierungs-Speichern zu kommunizieren und folglich die Benutzerauthentifizierungen für den ausgehenden Datenverkehr an Applikationen zu übernehmen. Darüber hinaus sind sie in der Lage die zugewiesenen Bandbreiten für Applikationen zu steuern (Russel, 2018).

Des Weiteren sind NGFWs in der Lage, den eigentlichen Datenstrom zu analysieren und ungewöhnliche Verhaltensmuster zu erkennen. Sie kombinieren den Paketfilter regulärer Firewalls eng mit einem IPS und können Aktivitäten der im Netzwerk befindlichen Benutzerinnen und Benutzer erkennen und auf Basis von Richtlinien steuern. Darüber hinaus können sie ebenfalls Antivirus- und Antispam Funktionen, das Filtern von Inhalten, VPN-Funktionen, URL-Filterung oder auch Advanced Threat Detection[4] übernehmen (Güttich & Schmitz, 2017).

Aufbauend auf diesen Prinzipien von Netzwerk- und Next Generation Firewalls haben sich mittlerweile ebenfalls noch sogenannte Web Application Firewalls am Markt etabliert. Solche Web Application Firewalls fokussieren sich auf die siebente Schicht des OSI-Modells (vergleiche Abbildung 3.3), den Application-Layer und werden in erster Linie eingesetzt, um Web-Server vor einer Vielzahl an Angriffsvektoren zu schützen. Die Verwendung von Access Control Lists (ACL), IP-Tables und die Analyse des Datenstroms sind dabei nur einige Funktionen, die eine solche Firewall für den Schutz von Web-Servern und darauf laufenden Applikationen, leistet (Endraca, King, Nodalo, Maria & Sabas, 2013).

[4] Erkennung von Angriffen, die mit fortschrittlicher Malware und dauerhaftem Fernzugriff versuchen, sensible Unternehmensdaten über einen längeren Zeitraum zu stehlen (Lord, 2018).

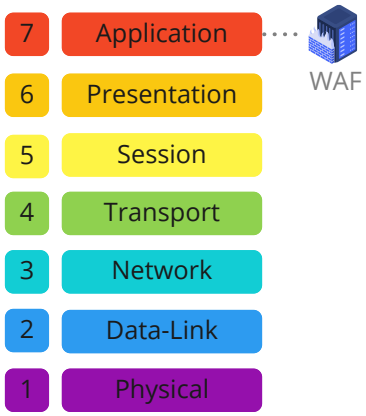


Abbildung 3.3: OSI-Schichten[5]

In diesem Kapitel wird der Fokus auf solche WAFs gelegt und deren Funktionsweise tiefgreifend erläutert.

3.1 Anwendungsfälle und Verwendung

Wie bereits erwähnt, werden Web Application Firewalls verwendet, um Web-Server und darauf laufende Applikationen auf Anwendungsebene vor einer Vielzahl an Bedrohungen zu schützen. WAFs werden typischerweise in Cloud-Umgebungen oder in demilitarisierten Zonen (DMZ)[6] eingesetzt, um den eingehenden Datenverkehr zu prüfen und somit Bedrohungen frühzeitig zu erkennen und zu beseitigen. Sie verfügen über fortschrittliche Erkennungs-Mechanismen, die sowohl Schutz vor Zero-Day-Angriffen[7], als auch Schutz vor den zehn häufigsten Schwachstellen der OWASP Foundation und den 25 häufigsten Softwarefehlern des SANS Instituts bieten (Russel, 2018).

Die zehn häufigsten Sicherheitslücken der OWASP Foundation aus dem Jahre 2021 sind in Abbildung 3.4 dargestellt.

[5] Grafik vom Autor erstellt.
[6] Neutrale Zone zwischen dem privaten und öffentlichen Netzwerk,welche die Beiden durch Firewalls und strenge Kommunikationsregeln voneinander trennt (Luber & Schmitz, 2018a).
[7] Ausnutzen von Sicherheitslücken, die dem jeweiligen Unternehmen zum Zeitpunkt noch unbekannt sind (Freda, 2021).

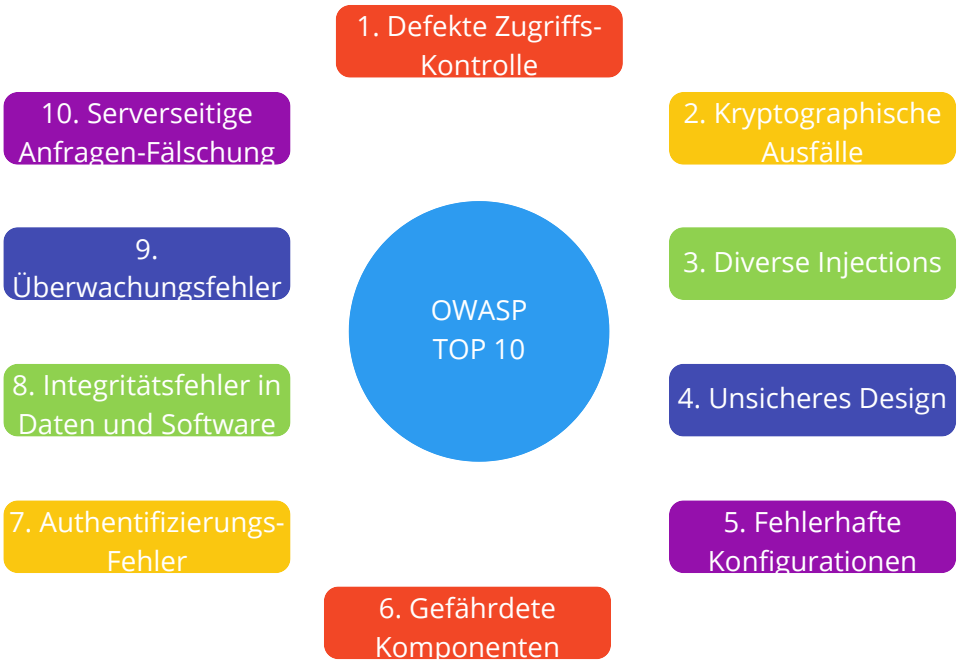


Abbildung 3.4: OWASP Top 10[8]

Einerseits bieten WAF einen Basisschutz vor bekannten Angriffen oder Schwachstellen durch Verwendung von sogenannten Blacklists. Das macht sie zu einem geeigneten Werkzeug, um industrielle Standards umzusetzen und den Anforderungen von gesetzlichen Regelungen gerecht zu werden. Besonders relevant wird der Einsatz von WAF allerdings beim Aufdecken von Sicherheitslücken durch Penetrationstests oder Code-Reviews. Auch wenn die Sicherheitslücke innerhalb der Anwendung zeitnah und mit vertretbarem Aufwand geschlossen werden kann, kann eine WAF bis zum tatsächlichen Ausrollen des Software-Patches die Lücke wesentlich schneller schließen. Dies tritt oft bei Systemen auf, welche aus mehreren Komponenten oder gar Fremdsystemen bestehen, deren Wartungszyklen sehr lange oder undefiniert sind (Dermann et al., 2008).

Über den Schutz vor bekannten Angriffen hinaus, besteht einer der Hauptnutzen für die Verwendung von Web Application Firewalls darin, bereits fertiggestellte Applikationen in Produktivumgebungen auf Anwendungsebene mit überschaubarem Aufwand und ohne Änderung an den Anwendungen an sich, nachträglich zu schützen. Das bietet sich sowohl für veraltete Software als auch für Fremdsysteme an, bei welchen eine Änderung des Source-Codes nur eingeschränkt beziehungsweise gar nicht möglich ist (Dermann et al., 2008).

Neben der schnellen Reaktionsfähigkeit in Bezug auf Sicherheitslücken, vereinfachen

[8]Grafik adaptiert von (OWASP Foundation, 2021a).

Web Application Firewalls ebenfalls den Prozess der Fehlerlokalisierung erheblich, vorausgesetzt die Implementierung der WAF bietet die Möglichkeit von zentralen Fehlermeldungen. Bei einem System, das aus mehreren Komponenten besteht, kann die Auswertung im Fehlerfall einige Zeit in Anspruch nehmen. Dies wird durch das Bereitstellen von zentralen Fehlermeldungen durch eine WAF erleichtert und beschleunigt die Fehlersuche. Aufbauend darauf kann eine WAF ebenfalls als zentrale Stelle für Monitoring und Reporting verwendet werden (Dermann et al., 2008).

Weiters definiert die OWASP Foundation folgende Vorteile für die Verwendung von Web Application Firewalls (Dermann et al., 2008):

- Session-Management
- URL-Verschlüsselung
- Site-Usage-Enforcement[9]
- Robustheit gegenüber externen Angriffen
- Lastenverteilung
- Terminierung von SSL-Verbindungen

3.1.1 Schutz vor SQL-Injection

Eine Funktion von Web Application Firewalls ist das Abwehren von SQL-Injection. SQL-Injection ist Teil der häufigsten Bedrohungen der OWASP Foundation und wurde bereits in Kapitel 2.1.2 genauer beschrieben. Diese Form des Angriffs basiert häufig auf Basis von HTTP-Post Anfragen und versucht manipulierten SQL Code an die der Applikation angeknüpfte Datenbank zu senden (Russel, 2018).

Es stellt sich demnach die Frage, wie eine Web Application Firewall diese Form des Angriffs erkennen und blockieren kann? Im Prinzip, nicht aber in der tatsächlichen technischen Umsetzung, ist das relativ einfach. Die WAF analysiert auf Basis von gewissen Regelsätzen die Eingabe, also die SQL-Abfrage, und sucht darin nach gewissen Merkmalen von typischen SQL-Injection Angriffen, beispielsweise nach reservierten Zeichen. Solche Regelsätze suchen oftmals nach Tautologien. Eine Tautologie bezeichnet eine

[9] Die mögliche Reihenfolge beim Aufrufen von URLs kann festgelegt oder erkannt werden (Dermann et al., 2008).

Bedingung, welche immer wahr ist. Das wohl einfachste Beispiel für eine Tautologie ist folgendes (Russel, 2018):

' or 1=1 - - '

Hierbei wird angenommen, dass es vor dem ersten einfachen Anführungszeichen bereits eine SQL-Abfrage gab. Durch diese Tautologie wird diese Abfrage um den Ausdruck *or 1=1* erweitert. Mit den zwei darauffolgenden Subtraktions-Operatoren, welche in SQL als Indikator für Kommentare verwendet werden, wird jedes weitere darauf folgende SQL-Statement als Kommentar markiert und vor einer Auswertung ausgeschlossen. Demnach würde die Auswertung dieser Abfrage in jedem Fall erfolgreich sein, da *Eins gleich Eins* in jedem Fall wahr ist, auch wenn das davorstehende Statement als „falsch“ ausgewertet werden würde. Eine WAF wirkt dem entgegen, indem sie solche Tautologien, unabhängig der verwendeten Werte, erkennt und in weitere Folge filtern und blockieren kann (Russel, 2018).

3.1.2 Schutz vor XSS

Neben der Vorbeugung vor SQL-Injection, kann eine Web Application Firewall ebenfalls Angriffe auf Basis des in Kapitel 2.1.2 beschriebenen Cross-Site-Scripting erkennen. XSS ist ebenfalls ein dominierender Eintrag in den OWASP Top 10 und bezeichnet das Einschleusen von manipuliertem HTML oder JavaScript Code in Webseiten und den dahinterliegenden Datenbanken (Russel, 2018).

Ähnlich wie eine WAF SQL-Injection vorbeugt, agiert diese auch bei XSS. Aufgrund der endlosen Möglichkeiten von manipulierten HTML oder JavaScript, ist es wenig sinnvoll, auf exakte Datensatzübereinstimmungen zu prüfen. Vielmehr müssen auch bei solchen Attacken gewisse Muster und Verhaltensweisen erkannt werden, um diese effektiv filtern oder blockieren zu können. Die Regelsätze von Web Application Firewalls sind bereits so vorkonfiguriert, dass sie bekannte Muster von XSS-Attacken innerhalb der URLs oder in den Textkörpern der Anfrage erkennen und blockieren können (Russel, 2018).

3.1.3 Schutz vor Session Tampering

Eine weitere bekannte Angriffsart, die sich innerhalb der 10 Gruppen der OWASP Foundation einbettet, ist die Manipulation von Sitzungen (Session Tampering, oder

Session Hijacking). Unter Session Tampering versteht man Angriffe, welche durch gezielte Manipulation von Sitzungsdaten wie Session-IDs, Session-Parameter oder auch Cookies, versuchen, die aktuelle Session von Benutzerinnen oder Benutzern zu übernehmen oder in deren Namen eine neue zu starten. Hierbei spricht man auch von Man-in-the-Middle oder Session-Replay Attacken (Russel, 2018).

Da bei der Kommunikation über das HTTP-Protokoll meist viele verschiedene TCP-Verbindungen verwendet werden, benötigen die Web-Server Methoden, um die Verbindung einzelner Benutzer zu erkennen. Dies wird oft durch einen Token sichergestellt, der nach erfolgreicher Authentifizierung an den Browser des Clients gesendet wird. Solche Tokens können in die URL eingebettet werden, im Body oder aber auch als Cookie in den HTTP-Headers. Beim Session-Hijacking wird versucht, solche Tokens zu stehlen oder zu manipulieren, um Zugriff auf den Web-Server zu erhalten. Dies kann beispielsweise durch Session-Sniffing, clientseitige Angriffe wie XSS oder auch durch Man-in-the-Middle-Angriffe geschehen (OWASP Foundation, 2021b).

Einige Implementierungen von Web Application Firewalls bieten die Möglichkeit, Artefakte wie beispielsweise Cookies, digital zu signieren, bevor diese an die Browser der Benutzerinnen und Benutzer weitergeleitet werden. Sendet der jeweilige Client diese signierten Cookies mit der nächsten Anfrage wieder retour, kann die WAF die Signatur der Cookies auslesen und entsprechend validieren. Durch dieses Prinzip kann ebenfalls sichergestellt werden, dass keine Fälschungen von Cookies in das System gelassen werden (Russel, 2018).

Eine weitere Möglichkeit in diesem Kontext ist es, dass WAF ebenfalls restriktieren, dass Clients nur mit sicheren Servern kommunizieren, welche über gültige Zertifikate verfügen. Somit kann sichergestellt werden, dass der Datenverkehr während der Kommunikation verschlüsselt ist, und die Web Application Firewall die übertragenen Daten entschlüsseln und in weiterer Folge vollständig prüfen kann (Russel, 2018).

3.2 Architektur

Wie bereits erwähnt, basieren Web Application Firewalls auf (vor-)definieren Regelwerken, auf Basis welcher der ein- oder ausgehende Traffic entweder weitergeleitet oder blockiert wird. Diese definierten Regelwerke teilen der WAF mit, nach welchen Schwachstellen, Sicherheitslücken und verdächtigem Traffic sie suchen müssen und was im Falle des Verletzens einer dieser Regeln passieren soll. Intelligente Implementierungen können die Quelle der Anfrage sogar herausfordern, um zu beweisen, dass sie von Menschen und nicht von Bots stammt. Wird eine Sicherheitslücke entdeckt, wird diese umgehend geschlossen, um böswillige Akteure daran zu hindern, diese auszunutzen (Sundar, 2021).

Sie können entweder als Hardware-Geräte bereitgestellt werden, als Software-Produkte (on-Premise oder in der Cloud) oder gar als eine Kombination aus beiden Konzepten und sind in der Regel auf Basis der folgenden Sicherheitsmodelle konfiguriert (Sundar, 2021):

- Whitelisting
- Blacklisting
- Hybrid

3.2.1 Sicherheitsmodelle & Regelwerke

Das Prinzip von **Whitelisting** basiert auf dem Grundkonzept, dass es wesentlich einfacher ist, zu definieren, welche Art von Anfragen gut und sicher für die dahinterliegenden Applikationen sind. Diese guten und sicheren Anfragen sind in der Regel wesentlich überschaubarer als jegliche Form von verdächtigen und schädlichen Angriffen, welche im Vergleich eine viel breitere Masse einnehmen und regelmäßig gepflegt und überarbeitet werden müssen (Haltdos Inc., 2020).

Artefakte wie Cookies, Routen und Parameter sind dem Unternehmen meist bekannt und können ohne viel Aufwand in eine solche Whitelist eingepflegt werden. Whitelisting ist demnach für Situationen geeignet, in welchen die Eingaben von ein- oder ausgehenden Anfragen bekannt sind und es schützt zukunftssicher vor unbekannten Bedrohungen (Haltdos Inc., 2020). Es wird empfohlen, im Raum von internen Netzwerken auf whitelist-basierte Web Application Firewalls zu setzen (Sundar, 2021).

Im Vergleich dazu, ist die Verwendung von **Blacklists** wesentlich aufwendiger und auch unsicherer. Bei der stetig wachsenden Menge an Gefahren, die in der heutigen Zeit auf Unternehmen und deren Benutzerinnen und Benutzer lauern, scheint es fast unmöglich, jede Form von bekannten Angriffsvektoren in einer Blacklist zusammenzufassen. Durch die immense Anzahl der Regeln, die entstehen würde, wenn die Web Application Firewall rein auf Basis von Blacklists konfiguriert wäre, leidet auch die Effizienz des Systems, da jedes ein- oder ausgehende Datenpaket die Validierung jeder dieser Regeln durchlaufen müsste (Haltdos Inc., 2020).

Darüber hinaus lässt sich auf Basis von Blacklist nur die Vergangenheit abbilden und es kann nicht auf neuartige, unerkannte Bedrohungen reagiert werden (Haltdos Inc., 2020). Die Verwendung von Blacklists wird für Applikationen im öffentlichen Raum empfohlen, da beispielsweise Distributed Denial of Service Attacken, ausgelöst durch eine große Anzahl an Anfragen derselben IP-Adresse, verhindert werden können (Sundar, 2021).

Wie der Name bereits erwarten lässt, wird im **Hybriden** Modell sowohl auf die Verwendung von Whitelists, als auch auf die Verwendung von Blacklists gesetzt. Dieses Konzept erlaubt es, speziell auf die Bedürfnisse von einzelnen Web-Applikationen im Detail einzugehen. Die Verwendung des hybriden Konzepts bietet sich sowohl für die Verwendung in privaten Netzwerken an, als auch für Netzwerke im öffentlichen Raum (Sundar, 2021).

Die Regelwerke, die auf Basis dieser Konzepte erstellt werden, um den tatsächlichen Inhalt der Datenpakete zu validieren, können dabei sehr einfach oder auch entsprechend komplex sein, je nachdem welche Bedrohungen von der jeweiligen Regel behandelt werden soll. Grundsätzlich besteht jede Regel aus drei Kernkomponenten (Smith, 2021):

- Objektmodell der HTTP-Anfrage
- Regular Expression Engine
- Punktesystem (Scoring-System)

Das **Objektmodell der HTTP-Anfrage** enthält die tatsächlichen Daten, die für die Validierung zur Verfügung stehen, wie beispielsweise die IP-Informationen, jegliche HTTP-Header, der Body der HTTP-Anfrage, der URL und entsprechende Variablen der Anfrage. All diese Informationen bilden eine Baum-Struktur, in welcher an verschiedenen Stellen nach unterschiedlichen Arten von Attacken gesucht werden kann (Smith, 2021). Dieses Objektmodell ist in Abbildung 3.5 exemplarisch dargestellt.

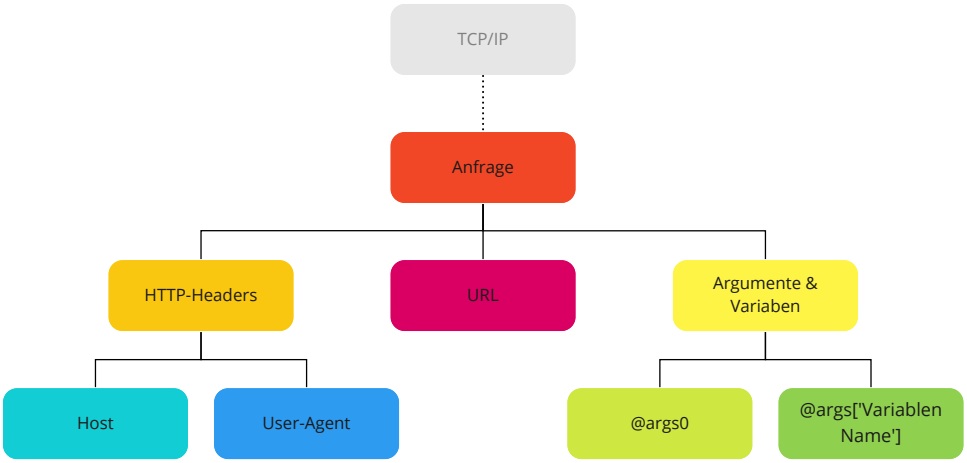


Abbildung 3.5: Objektmodell eines HTTP Requests[10]

Die zweite Komponente bildet eine sogenannte **Regular Expression Engine**. Nachdem so gut wie alle Implementierungen von Web Application Firewalls auf Basis von gewissen Mustern nach Angriffs-Indikatoren suchen, bildet eine solche Engine einen äußerst wichtigen Teil eines WAF Systems. Diese ist demnach dafür verantwortlich, diese definierten Muster in den empfangenen Daten zu erkennen (Smith, 2021).

Die dritte und letzte Komponente bildet ein **Scoring-System**. Dieses Punktesystem ist dafür zuständig, um zu validieren, ob eine Bedrohung vorhanden ist oder nicht. Das Scoring-System entscheidet, ob genug Treffer einer Regel angeschlagen haben, um die Anfrage als Bedrohung ansehen zu können (Smith, 2021).

Diese drei Komponenten bilden demnach das Herzstück von Web Application Firewalls, um das regelbasierte Blockieren von ein- und ausgehenden HTTP-Anfragen gewährleisten zu können. Einfache Regeln zielen in den meisten Fällen auf die Metadaten der Anfragen ab und limitieren diese auf Basis simpler Kriterien. Einige Beispiele für solche einfachen Regeln sind (Smith, 2021):

- Der URL darf nicht länger als 500 Zeichen sein
- Es muss ein Host-Header angegeben sein
- Der User-Agent Header darf das Wort „Bot“ nicht enthalten
- Das verwendete HTTP-Protokoll muss mindestens der Version HTTP/1.1 entsprechen

Wesentlich komplexer werden diese Regelwerke, wenn versucht wird Angriffe wie

[10] Grafik adaptiert von (Smith, 2021).

Command-, SQL-Injection oder Cross-Site-Scripting zu erkennen. Die Ursache dafür liegt darin, dass dabei Rücksicht auf Sonderzeichen und reservierte Schlüsselwörter genommen werden muss, diese Schlüsselwörter sich allerdings mit normalen, harmlosen Daten überschneiden können. Beispielsweise können Wörter wie „SELECT“, „TABLE“ oder „USER“, welche reservierte Ausdrücke im SQL-Dialekt darstellen, ebenfalls in harmlosen Daten, wie etwa Blogposts vorkommen (Smith, 2021).

Das Scoring-System einer WAF versuch dem mithilfe eines Punktesystems und definierten Schwellwerten entgegenzuwirken. Dabei werden Sonderzeichen und Schlüsselwörter an eine Punktezahl geknüpft. Beispielsweise bekommen reservierte Sonderzeichen von SQL-Statements zehn Punkte zugewiesen, während SQL-Schlüsselbegriffe fünf Punkte zugewiesen bekommen. Ausdrücke, die nicht aus einem reservierten Bereich stammen, werden mit null Punkten quantifiziert. Jeder Treffer addiert nun die definierte Punktezahl zum Ergebnis hinzu. Übersteigt das Ergebnis schlussendlich den definierten Schwellwert, wird von einer Attacke ausgegangen (Smith, 2021).

Ein einfaches Beispiel für ein Scoring System zur Erkennung von SQL-Injection wird von Smith (2021) genannt und illustriert. Geht man von folgendem SQL-Injection Angriff aus

Robert');DROP TABLE STUDENTS;- -

kann der Inhalt in unterschiedliche Teile aufgeteilt werden: Reservierte Zeichen, Schlüsselwörter und übrig bleibende Ausdrücke. Dies ist exemplarisch in einem Venn-Diagramm in Abbildung 3.6 dargestellt.

Quantifiziert man nun das angeführte Beispiel mit den davor festgelegten Punktewerten, ergibt sich das Ergebnis aus Tabelle 3.1. Liegt das Ergebnis dieser Regel nun unter dem definierten Schwellenwert, wird nicht von einem Angriff ausgegangen. Liegt es darüber, schlägt die WAF Angriffsalarm, und blockiert die entsprechenden Datenpakete. Diese Quantifizierung kann dabei recht flexibel und modular sein. Beispielsweise kann es unterschiedliche Schwellwerte für die unterschiedlichen zu erkennenden Bedrohungen geben. Allerdings kann so ein Schwellwert ebenfalls für die gesamte Anfrage definiert werden und nicht nur für einzelne Teile des HTTP-Objektmodels (Smith, 2021).

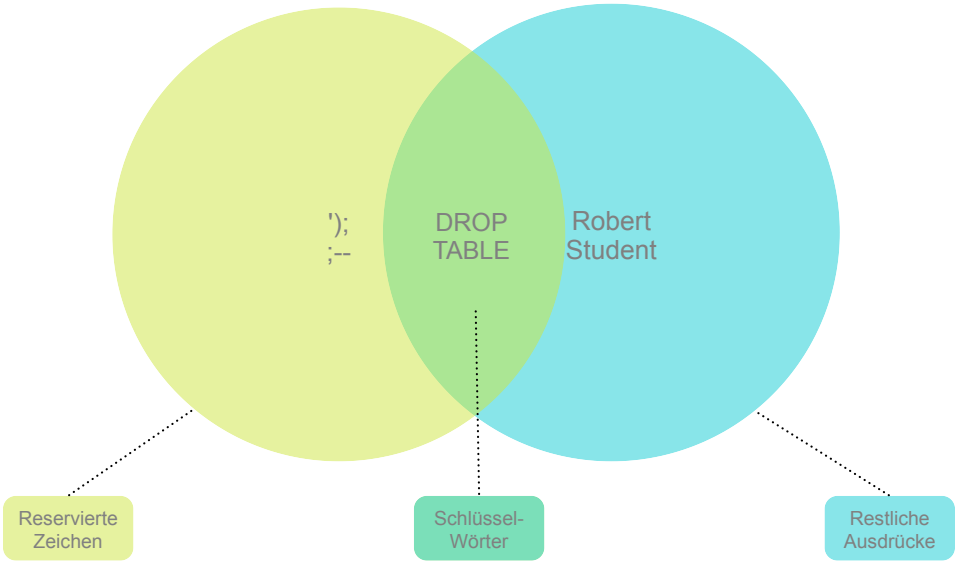


Abbildung 3.6: Teile einer SQL-Injection[11]

Wert	Punkte
Robert	0
’);	10
DROP	5
TABLE	5
Students	0
;- -	10
Summe:	**30**

Tabelle 3.1: Quantifizierung SQL-Injection[12]

Demnach erschweren solche komplexen Regeln nicht nur die korrekte Konfiguration, sondern führen auch häufig zu erhöhten Falsch-Positiv-Raten (false-positiv rates), also zu fälschlicherweise als Bedrohung eingestuften Anfragen, die in der Realität gar keine sind. Dies stellt eine Herausforderung für den effizienten und korrekten Betrieb einer Web Application Firewall dar. Ziel bei der Verwendung von WAF sollte sein, diese Falsch-Positiv-Raten möglichst gering zu halten (Smith, 2021).

Gleichzeitig muss aber auch versucht werden, die Falsch-Negativ-Raten (false-negativ rates) auf einem Minimum zu halten, also Anfragen, die fälschlicherweise nicht als Bedrohung angesehen wurden, obwohl sie eine darstellen. Die meisten Administratoren und Sicherheitsbeauftragten verbringen circa 80% ihrer Zeit mit dem Versuch, ihre Regelwerke auf das akkurate Niveau zu heben, wie es von den dahinterliegenden Applikationen verlangt wird, und die Raten entsprechend gering zu halten (Smith, 2021).

[11] Grafik adaptiert von (Smith, 2021).
[12] Daten übernommen von (Smith, 2021).

3.2.2 Typen & Betriebsarten

Für die Bereitstellung (Deployment) und das tatsächliche Einsatzgebiet von Web Application Firewalls gibt es unterschiedliche Ansätze und Möglichkeiten, auf welche je nach Anforderung zurückgegriffen werden kann. Die Einsatzgebiete reichen dabei von Server-Plugins, über Filter-Installationen bis hin zu maßgeschneiderten Lösungen für spezielle Applikationen. Darüber hinaus kann zwischen folgenden unterschiedliche Arten von Web Application Firewalls unterschieden werden (Sermpinis, 2017):

- Hardware-basierte WAF
- Cloud-basierte und hybride WAF
- Code-integrierte WAF

Hardware-basierte Web Application Firewalls bezeichnen ein Stück Hardware, das physisch auf einem Server installiert wird. Damit werden sie vor der Web-Infrastruktur des Servers platziert und ermöglichen, dass für den WAF Service keine Verbindung zu einem Remote-Server hergestellt werden muss. Nachdem hardware-basierte Lösungen vor jeglicher Hardware der eigentlichen Web-Server platziert werden, muss der gesamte Datenverkehr initial die WAF passieren, bevor er an die entsprechende Web-Applikation weitergeleitet wird. Diese Art von WAF ist äußerst zuverlässig und befindet sich in den eigenen vier Räumen, wodurch das Beheben von Problemen vereinfacht wird. Gleichzeitig kann dies als Nachteil angesehen werden, da bei auftretenden Problemen Fachpersonal vor Ort sein muss (Sermpinis, 2017).

Darüber hinaus muss diese Form von Web Application Firewall mit dem gesamten Datenverkehr und den Anforderungen unterschiedlicher Protokolle des Servers zurechtkommen und somit für jeglichen denkbaren Fall konfiguriert werden, um zukünftig keine Probleme zu verursachen (Sermpinis, 2017).

Cloud-basierte (oder hybride) WAF Systeme hingegen, werden extern auf entfernten Servern von Cloud-Dienstleistern oder in Zusammenarbeit mit solchen betrieben. Dieser Typ von WAF kann aus einer Kombination aus Hardware und Remote-Cloud-Services bestehen, bei welcher der Traffic initial im Remote-Cloud-Service gefiltert wird, bevor dieser an den jeweiligen Web-Server weitergeleitet wird. Solche Web Application Firewalls bieten in den meisten Fällen einen ausgezeichneten Schutz vor Distributed Denial of Service Angriffen, welche sich im letzten Jahrzehnt zu einer der häufigsten Bedrohungen für Unternehmen etabliert hat (Sermpinis, 2017).

Abgesehen vom Schutz vor DDoS Angriffen, bietet diese Art von Web Application Firewalls einen Mehrwert für Unternehmen, welche ihren Betrieb über mehrere verteilte Standorte hinweg ausführen und nicht jeder dieser Standorte über eine eigene physische Server-Infrastruktur verfügt oder eine eigene Lösung benötigt (Sermpinis, 2017).

Neben den bereits genannten Lösungen gibt es ebenfalls noch **Software- oder Code-integrierte** Web Application Firewall Systeme. Diese basieren rein auf Softwareimplementierungen und können entweder direkt in die zu schützenden Web-Applikationen integriert werden oder als Softwareinstallation am Server bereitgestellt werden. Dabei muss der Datenverkehr zuerst die zur Filterung verwendete Software passieren, bevor dieser den eigentlichen Server oder die Ziel-Applikation erreichen kann (Sermpinis, 2017).

Eine softwareseitige Lösung hat den Vorteil, dass diese relativ kosteneffizient ist, die Problembehandlung leichter fällt und wesentlich leichter aktualisiert werden kann. Zwar wird für solche Lösungen teilweise trotzdem spezielle Hardware benötigt, allerdings kann die Wartung einer softwareseitigen Lösung auch vom regulären IT-Personal übernommen werden (Sermpinis, 2017).

Neben den unterschiedlichen Typen von Web Application Firewalls, gibt es ebenfalls noch unterschiedliche Arten, um diese zu betreiben. Einige unterschiedliche Möglichkeiten für den Betrieb von WAF umfassen folgende (Sermpinis, 2017; Russel, 2018; Anders, o. J.):

- Reverse Proxy
- Transparent Proxy
- Forward Proxy
- Netzwerk-Überwachungsmodus
- Server-Modus

Reverse Proxy

Die meisten Menschen betrachten eine Web Application Firewall als **Reverse-Proxy-Server**. Diese Art des Deployments leitet eingehenden Datenverkehr auf Basis von Network Address Translation (NAT)[13] zwischen internen und externen Netzwerken

[13] Ersetzen von Quell- oder Ziel-IP-Adressen von Datenpaketen durch andere Adressen (Luber & Donner, 2018).

an die Ziel-Server weiter. Die WAF besitzt dabei eine eigene IP-Adresse (Sermpinis, 2017). NAT sorgt in diesem Kontext dafür, dass die internen IP-Adressen vollständig vor der Außenwelt verborgen werden. Wird eine WAF als Reverse-Proxy eingesetzt, läuft der gesamte Datenverkehr über die WAF und es gibt keine Datenpakete, die daran vorbeikommen (Russel, 2018).

Eine Ausführung dieser Variante bildet das Dreibeinige-Modell. In diesem Modell wird nur eine einzige WAF eingesetzt, welche über drei getrennte Schnittstellen verfügt. Eine öffentliche Schnittstelle, eine interne und eine abgeschirmte für Subnetze, wie beispielsweise eine DMZ. Dieses dreibeinige Modell hat den Vorteil, dass wenig Hardware benötigt wird, bringt aber simultan die Nachteile, dass keine hohe Verfügbarkeit erreicht werden kann und, dass es bei falscher Konfiguration zur Gefährdung des internen oder abgeschirmten Netzwerkes kommen kann (Russel, 2018). Dieses Modell wird exemplarisch in Abbildung 3.7 dargestellt.

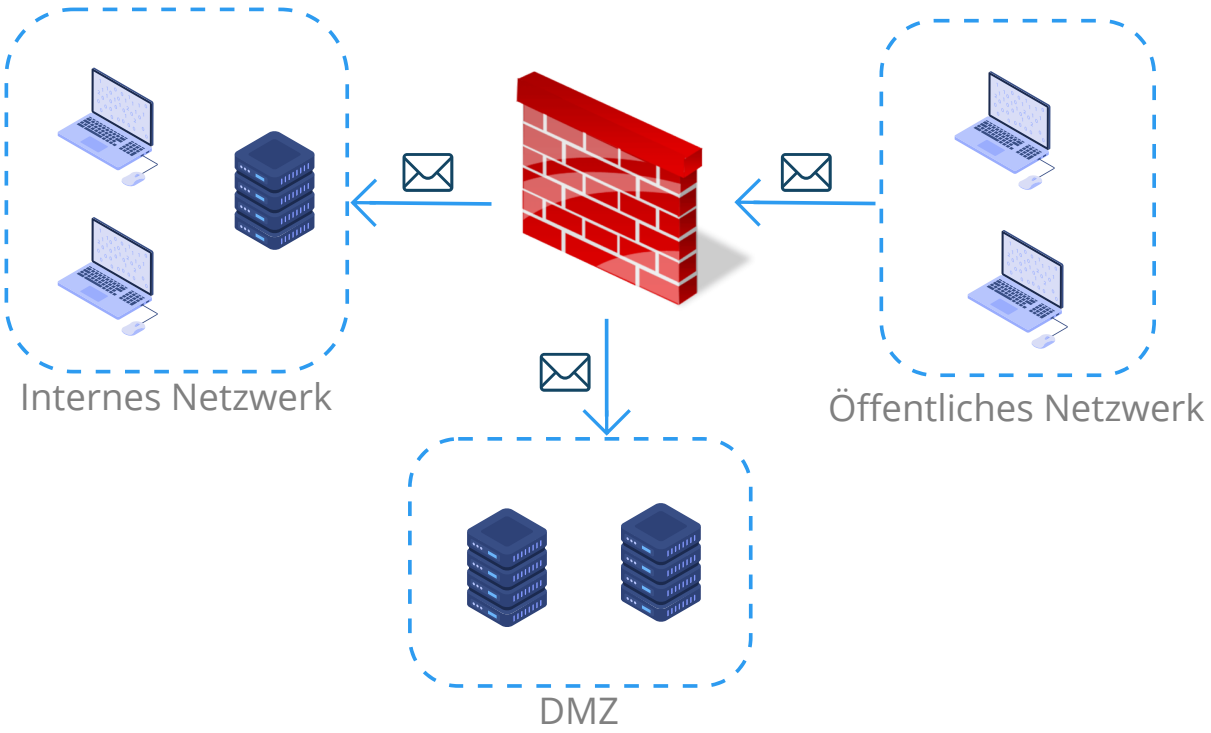


Abbildung 3.7: Dreibeinige WAF[14]

Bewährter ist allerdings die Bereitstellung von WAF als sogenanntes Firewall-Sandwich, wie es in 3.8 dargestellt ist. In diesem Modell setzt man auf den Einsatz von zwei WAF. Die erste befindet sich am Rand des äußeren Netzwerkes und die zweite am Rand des Inneren. Der Bereich zwischen den beiden WAF fungiert als DMZ. Ein Firewall-Sandwich bringt den Vorteil, dass die Änderung einer Firewall nicht zwanghaft das interne Netzwerk

[14] Grafik vom Autor erstellt.

gefährdet. Darüber hinaus kann in diesem Modell die äußerste WAF als SSL/TLS-Terminierungspunkt verwendet werden. Durch diese Terminierung hat die zweite (innere) WAF die Möglichkeit, den Inhalt der Datenpakete genauer zu betrachten (Russel, 2018).

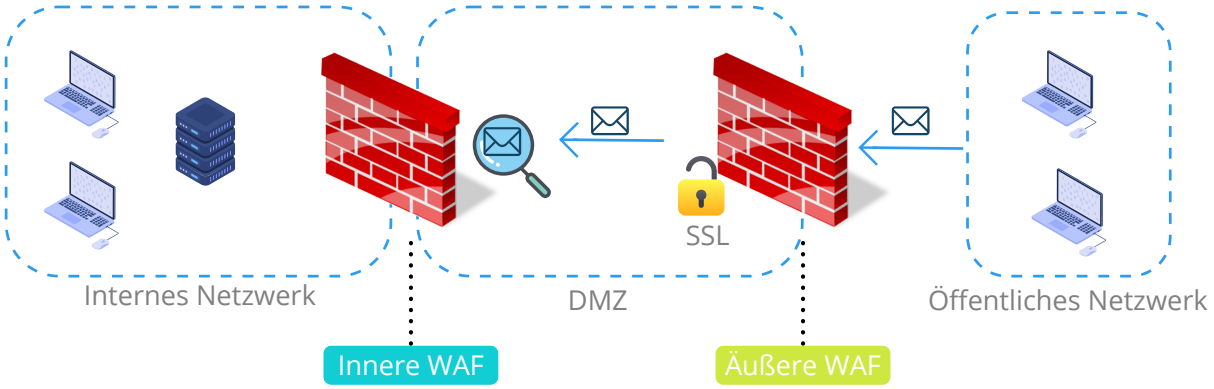


Abbildung 3.8: WAF Sandwich[15]

Transparent Proxy

Ähnlich der Bereitstellung als Reverse-Proxy ist die Verwendung als **Transparent Proxy**, dargestellt in Abbildung 3.9. Der Unterschied zur Verwendung als Reverse-Proxy ist, dass die WAF in dieser Form keine eigene IP-Adresse besitzt und als transparenter Layer 2 Switch fungiert, da sie hinter einer Netzwerk-Firewall platziert wird. Die Verwendung als transparenter Proxy bringt die Vorteile, dass hohe Verfügbarkeit erreicht werden kann und keine Änderungen am Netzwerk oder den dahinterliegenden Applikationen notwendig sind (Russel, 2018; Sermpinis, 2017).

Einige Implementierungen von Web Application Firewalls unterstützen darüber hinaus einen „Routing-Mode“. Ein solcher Modus ist vorteilhaft, wenn NAT verwendet werden muss, oder wenn gewisse Subnetze in einen anderen IP-Adressbereich fallen als die restlichen Teile des Netzwerks (Russel, 2018).

[15] Grafik vom Autor erstellt.

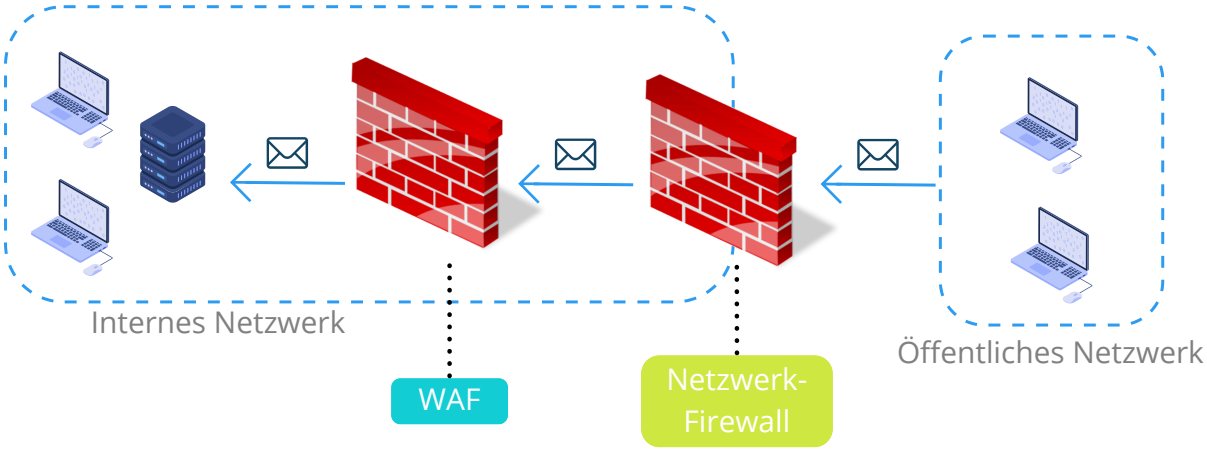


Abbildung 3.9: Transparent Proxy WAF[16]

Forward Proxy

Ein Forward-Proxy definiert das Gegenstück zu einem Reverse-Proxy. Dieser ist ebenfalls am Rande des Netzwerkes stationiert, regelt allerdings im Gegensatz zu einem Reverse-Proxy nur den aus dem Netzwerk ausgehenden Datenverkehr nach vorgegebenen Richtlinien. Solche Forward-Proxies werden in der Regel in den internen Netzwerken von großen Organisationen, wie Universitäten oder Unternehmen, betrieben. Dadurch sind diese in der Lage, das Verhalten der sich im Netzwerk befindlichen Benutzerinnen und Benutzer auf gewisse Art und Weise zu restriktieren und zu überwachen (Lakhno et al., 2022; Imperva Inc., o. J.; Bhardwaj, o. J.).

Einerseits können die Benutzerinnen und Benutzer daran gehindert werden, gewisse Webseiten zu besuchen. Darüber hinaus, kann das Verhalten dieser überwacht und verdächtiger Datenverkehr blockiert werden, bevor dieser einen Server erreicht. Abgesehen von Restriktionen gegenüber den Anwenderinnen und Anwendern, kann durch Verwendung eines Forward-Proxies gleichzeitig die Erfahrung und Sicherheit für diese verbessert werden. Dies wird ermöglicht, da der Inhalt von oft besuchten Webseiten in Caches gehalten und so die Antwortzeiten beim Besuch solcher erheblich beschleunigt werden kann. Darüber hinaus verschleiert der Forward Proxy die IP-Adresse der Clients und ermöglicht diesen somit einen anonymen Auftritt im Internet (Lakhno et al., 2022; Imperva Inc., o. J.; Bhardwaj, o. J.). Der Betrieb einer Web Application Firewall als Forward-Proxy ist in Abbildung 3.10 dargestellt.

[16] Grafik adaptiert von (Anders, o. J.).

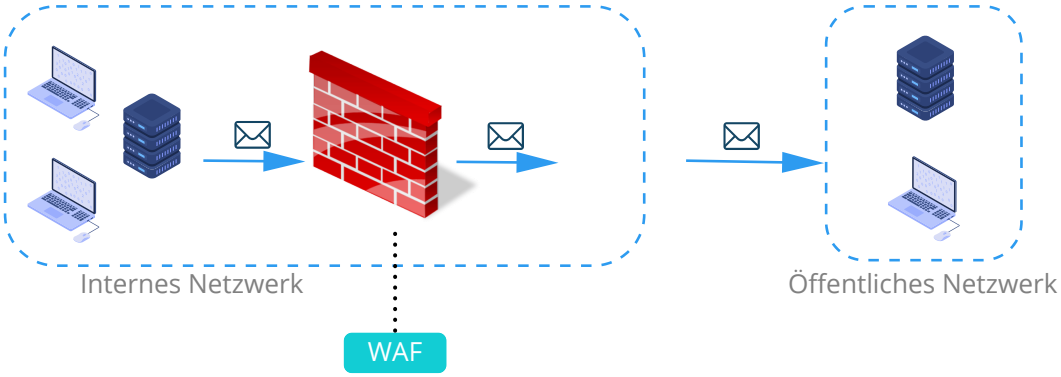


Abbildung 3.10: Forward Proxy WAF[17]

Netzwerk-Überwachungsmodus

Eine Web Application Firewall kann ebenfalls in einem Netzwerk-Überwachungsmodus betrieben werden, auch Out-Of-Band genannt, dargestellt in Abbildung 3.11. Die WAF wird dabei direkt vor dem Web-Server platziert und überwacht den eingehenden Datenverkehr des Servers über Test Access Points (TAP)[18] oder Switch Port Analyzer Ports (SPAN) [19].

Das bedeutet, dass die WAF nicht direkt innerhalb des Datenflusses platziert wird, sondern den Datenverkehr spiegelt und passiv analysiert. Dieser Modus bietet eine leichte und schnelle Bereitstellung des WAF Services, hat eine äußerst niedrige Latenz und bietet sich gut für Pilotprojekte und Tests an. Allerdings sind in diesem Modus nicht alle Funktionen der WAF verfügbar und ein Blockieren von verdächtigem Datenverkehr oder erkannten bedrohungen ist nur mithilfe von TCP-Resets[20] möglich (Sermpinis, 2017; Anders, o. J.; Russel, 2018).

[17] Grafik adaptiert von (Lakhno et al., 2022).
[18] Dediziertes Gerät zur passiven Aufteilung des Netzwerkverkehrs (Schubert, 2021).
[19] Dedizierter Port eines Netzwerk-Switch zur Erstellung von Kopien der Datenpakete und übermittlung dieser an angeschlossene Tools (Schubert, 2021).
[20] Senden eines bestimmten TCP Datenpakets, um auf Probleme hinzuweisen und ein Terminieren bzw. erneutes Aufbauen der Verbindung zu indizieren (Moles, 2021).

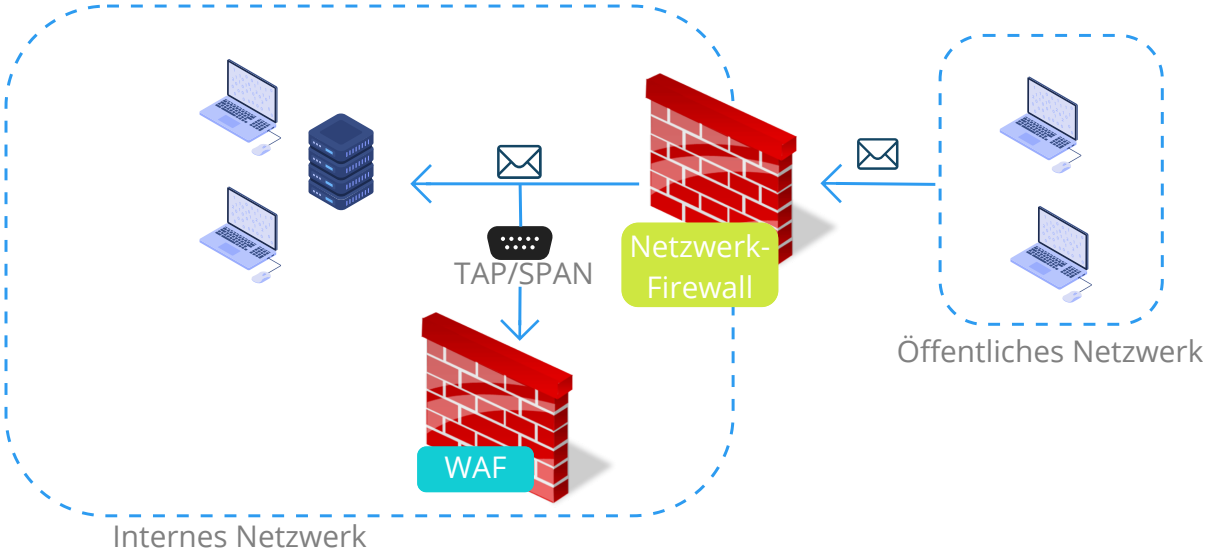


Abbildung 3.11: WAF Monitoring Mode[21]

Server-Modus

Im Server-Modus, abgebildet in Abbildung 3.12, befinden sich die WAF direkt an den zu schützenden Web-Servern. Dabei werden sie anhand von Plugins, Service-Modulen oder eigenständigen Software-Applikationen direkt am Server betrieben. Eine Web Application Firewall in diesem Modus ist ebenfalls leicht bereitzustellen, bringt aber den Nachteil, dass wichtige Ressourcen des Servers verbraucht werden. Darüber hinaus können auch in diesem Betriebsmodus nicht alle Funktionen voll ausgenutzt werden und Änderungen müssen streng kontrolliert werden, um sowohl die korrekte Funktionsweise der WAF, als auch die der Web-Server zu gewährleisten (Anders, o. J.).

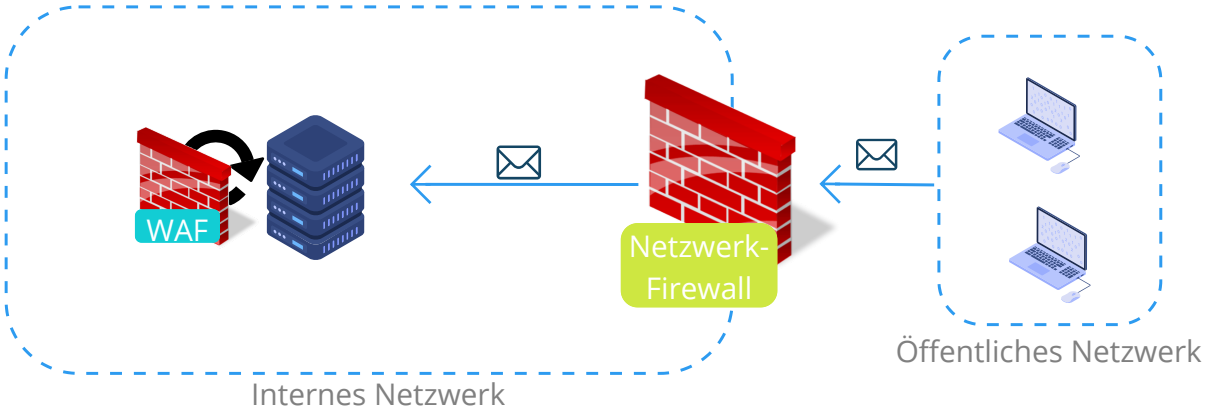


Abbildung 3.12: WAF Server Mode[22]

[21] Grafik adaptiert von (Anders, o. J.).
[22] Grafik adaptiert von (Anders, o. J.).

3.3 Grenzen & Herausforderungen

Obwohl Web Application Firewalls eine breite Palette an Funktionen bieten, um Applikationen vor bekannten Bedrohungen zu schützen, ist dies in den letzten Jahren wesentlich ineffizienter geworden, als es in der Vergangenheit war. Das Filtern, Überwachen und Blockieren von bösartigem Datenverkehr reicht heutzutage nicht mehr aus, um die Server und Daten eines Unternehmens effektiv vor Cyberangriffen zu schützen (L7 Defense Ltd., 2021).

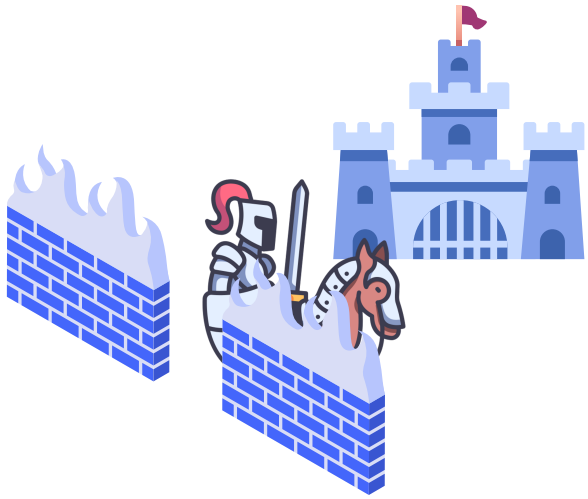


Abbildung 3.13: Der Ritter der Firewall: Durchbruch[23]

Der Grund dafür ist, dass moderne Angriffe immer komplexer werden und aus einer Kombination von technischen und funktionalen Angriffsformen bestehen. Während ein technischer Angriff beispielsweise versucht, Dienste durch DDoS-Attacken von Bot-Netzwerken zu unterbrechen, wird durch einen funktionalen Angriff versucht, Schwachstellen zur persönlichen Bereicherung auszunutzen oder die Geschäftslogik durch Manipulationen zu stören. Ersteres erfordert funktionsunabhängige- und richtlinienbasierte Schutzmaßnahmen, um solchen Angriffen effektiv vorbeugen zu können. Letzteres hingegen benötigt eine vollständige Nutzlastanalyse (L7 Defense Ltd., 2021).

Darüber hinaus geraten traditionelle Netzwerk-Umgebungen immer weiter in den Schatten, da Unternehmen immer mehr auf Cloud-native Infrastrukturen setzen und Applikationen, Abläufe und Daten in die Cloud verlagern. Daraus ergeben sich neue Vektoren für Angriffe, neue Sicherheitslücken und neue Herausforderungen, um sich effizient vor Bedrohungen zu schützen (L7 Defense Ltd., 2021).

[23] Grafik vom Autor erstellt.

3.3.1 Herausforderungen

Der Betrieb einer WAF unterstützt Unternehmen zwar dabei, die Sicherheit für ihre Anwendungen zu erhöhen, gleichzeitig aber ergeben sich dadurch gewisse Herausforderungen, die einen effizienten Einsatz erschweren. Babu (2019) fasst diese Herausforderungen in folgende fünf Punkten zusammen (Babu, 2019):

- Viele Alarme durch False-Positives
- Langwierige Untersuchungszeiten
- Unerkannte Angriffe durch False-Negatives
- Erhöhte Wartung durch DevOps
- Hohe Kosten

In Kapitel 3.2.1 wurde bereits ein kurzer Einblick in **False-Positive-Rates** gegeben. Diese False-Positives beschreiben das Blockieren von Anfragen, welche fälschlicherweise als Bedrohung eingestuft wurden. Je höher diese Rate beziehungsweise die Anzahl dieser False-Positives ist, desto mehr Warnungen werden von der WAF ausgegeben und legitimer Datenverkehr wird blockiert. Ist die Rate der False-Positives sehr hoch, können Administratorinnen und Administratoren der WAF leicht von der Anzahl an zu prüfenden Warnungen überwältigt werden (Babu, 2019; Wickett, 2018).

Nachdem WAF außerhalb der Anwendungen platziert sind und nicht wissen, für welche Bedrohungen die Anwendungen anfällig sind, ist es für die Web Application Firewall schwierig zu entscheiden, welche Angriffe echt sind und welche nicht. Hohe False-Positive-Rates lassen Unternehmen dazu neigen, ihre WAF nicht im Blocking-Mode (sondern nur im Monitoring-Mode) zu betreiben, oder ihre WAF-Regel auf einem Minimum zu halten, welche nur die offensichtlichsten Bedrohungen erkennen können (Babu, 2019; Wickett, 2018).

Darüber hinaus können solche False-Positives ebenfalls Auswirkungen auf die Benutzererfahrung der Kundinnen und Kunden des Unternehmens haben. Beispielsweise können potenzielle Kundinnen und Kunden ihre Bestellungen nicht platzieren, da diese fälschlicherweise als Bedrohung eingestuft und die entsprechende Anfrage von der WAF blockiert wurde (Babu, 2019; Wickett, 2018).

Einhergehend mit einer hohen False-Positive-Rate und entsprechend vielen Meldun-

gen der WAF, sind **langwierige Untersuchungszeiten** für die Administratorinnen und Administratoren der Web Application Firewall. Einerseits muss gerade beim Einführen einer WAF auf eine möglicherweise sehr hohe False-Positiv-Rate eingegangen werden, andererseits bedarf es laufend Anpassungen an den Regeln, um diese False-Positives zu eliminieren (Babu, 2019; Novikov, 2019).

Der Kontext, den eine WAF untersucht, hält sich dabei in Grenzen und geht meist nicht über die Daten einer HTTP-Anfrage hinaus. Für die Personen, welche die WAF administrieren, wird es aufgrund dieses geringen Kontexts erschwert, effektiv auf Vorfälle einzugehen genug Informationen zu sammeln, um die verdächtige Aktivität untersuchen zu können. Dadurch wird viel Zeit und Fachwissen vom zuständigen Personal erfordert, um eine WAF effektiv zu betreiben (Babu, 2019; Novikov, 2019).

Neben False-Positives erschweren auch **False-Negatives** den effizienten Betrieb einer WAF erheblich. Solche False-Negatives bezeichnen bösartige Anfragen, die von der WAF nicht als solche erkannt werden und demnach nicht blockiert werden können. Diese False-Negatives sind in der Regel Angriffe, welche sich nur schwer durch bekannte Muster erkennen lassen. Beispielsweise führen XML External Entity (XXE) Angriffe häufig zu solchen False-Negatives. Bei XXE Angriffen ist eine Referenz auf eine externe Entität innerhalb der übermittelten XML Daten eingebettet. Diese Entität ist daher nicht direkt Teil der übermittelten Daten, sondern als referenzierender URI integriert, der erst von der Applikation, die die Daten einliest, aufgelöst wird (Babu, 2019; OWASP Foundation, 2020).

Der WAF wird es somit unmöglich gemacht, verdächtige Muster innerhalb dieser externen Entität zu erkennen. Die Eliminierung von False-Negatives stellt daher eine schwierige Herausforderung für die administrierenden Personen der WAF dar, da diese das Eindringen von tatsächlich schadhaften Anfragen in das System des Unternehmens ermöglichen (Babu, 2019; OWASP Foundation, 2020).

Zusätzlich zur Herausforderung, Angriffe effektiv zu erkennen, erschweren ebenfalls moderne **DevOps-Prozesse** den effizienten Einsatz von Web Application Firewalls. WAF werden immer öfter in Umgebungen platziert, in welchen eine häufige Änderung des Software-Codes der dahinterliegenden Applikationen der Fall ist. Moderne Web Applikationen basieren auf Continuous Integration und Delivery und werden somit laufend mit Updates und Änderungen versorgt. Liegt eine solche Applikation hinter einer WAF, muss damit gerechnet werden, dass häufige Änderungen an der Applikation selbst, auch Änderungen an der WAF voraussetzen (Babu, 2019; Velasco, 2018).

Schlussendlich spielen auch Kosten eine wichtige Rolle für einen effizienten Einsatz von WAF. Gerade in Cloud-Umgebungen könne die Kosten sehr schnell sehr hoch werden. Besonders dann, wenn die WAF nicht für eine Bereitstellung in der Cloud ausgelegt ist. Zusätzlich muss darauf Rücksicht genommen werden, dass die Richtlinien der WAF angepasst werden müssen, wenn sich die dahinterliegende Applikation ändert oder skaliert wird. Somit sind laufend Anpassungen an der WAF selbst notwendig, die viel Zeit in Anspruch nehmen können und enstsprechend im Budget eingeplant werden sollten (Babu, 2019; Novikov, 2019).

Demnach gibt es beim Betrieb einer WAF einiges zu beachten und es sollte davon abgesehen werden, sich auf das reine Bereitstellen einer solchen zu verlassen. Vielmehr muss diese regelmäßig überarbeitet und verbessert werden, um den effektiven Schutz der dahinterliegen Applikationen zu gewährleisten. Allerdings gibt es ebenfalls einige Aspekte, die eine WAF nicht berücksichtigen kann.

3.3.2 Grenzen

Neben den genannten Herausforderungen beim Betrieb von WAF, gibt es ebenfalls Grenzen, welche eine WAF nicht überschreiten kann. Zwar wurden WAF in vielen Bereichen von Unternehmen adaptiert, trotzdem scheinen sie veraltet zu sein und sich nicht als bewährte Sicherheitsmethode etabliert zu haben (Spherical Defence AI Inc, 2018).

Eine Grenze von Web Application Firewalls ist der Umgang mit **Zero-Day-Exploits**. Eine WAF kann nur auf Basis der definierten Regeln agieren. Zero-Day-Exploits bezeichnen allerdings neue Sicherheitslücken, die zuvor noch nicht bekannt waren und für welche noch kein Patch verfügbar ist. Zwar kann anhand dieser Regeln betroffener Datenverkehr schnell blockiert werden, allerdings kann die WAF nur schwer selbst auf solche Bedrohungen reagieren (Spherical Defence AI Inc, 2018).

Zero-Day-Exploits aber auch Änderungen der Applikationen und das Entstehen neuer Sicherheitslücken bringen einen hohen **Wartungsaufwand** der WAFs mit sich. Die Wartung von WAF-Netzwerken ist äußerst zeitintensiv und muss regelmäßig gemacht werden. Jedes neue Feature einer zu schützenden Applikation und jede neue Sicherheitslücke müssen mit der WAF abgestimmt werden. Dies kann ein großes Problem darstellen. Anpassungen an der WAF können versäumt werden oder bestehende Regeln werden dahingehend geändert, dass sie ihren eigentlichen Zweck nicht mehr erfüllen und weitere Sicherheitslücken offenlegen (Spherical Defence AI Inc, 2018).

Des Weiteren haben WAF Systeme Probleme damit, **Replay-Attacken** zu erkennen. Die Funktionsweise von WAF schränkt diese auf die Analyse einzelner Anfragen ein und macht sie gegenüber Angriffen, die über mehrere Anfragen hinweg geschehen, machtlos. Replay-Attacken fallen beispielsweise in diese Kategorie. Solche Angriffe senden gültige Anfragen in böswilliger Absicht wiederholt oder verzögert (Spherical Defence AI Inc, 2018).

3.4 Anbieter

Über die Jahre hat sich eine Vielzahl an Anbietern von Web Application Firewall Lösungen etabliert. Diese reichen von namhaften Unternehmen, über Nischen-Spieler bis hin zu vollständigen Open-Source-Lösungen. Jeder dieser Anbieter hat dabei seine Stärken und Schwächen. In diesem Abschnitt soll etwas näher auf einige dieser Anbieter von kommerziellen als auch von Open-Source Lösungen eingegangen werden (D'Hoinne, Hils, Kaur & Watts, 2020).

3.4.1 Kommerziell

Das Marktforschungsunternehmen Gartner hat bereits mehrfach Berichte über Web Application Firewalls veröffentlicht. Unter diesen Berichten befinden sich unter anderem Analysen unterschiedlicher Anbieter von WAFs, welche die diversen Unternehmen auf Basis ihrer Stärken und Schwächen in folgende Kategorien gliedert (D'Hoinne et al., 2020):

- Nischen-Spieler
- Visionäre
- Herausforderer
- Führer

Die Analyse von Gartner aus dem Jahr 2020 listet zehn verschiedene Anbieter von WAF-Systemen auf und weist ihnen jeweils eine der genannten Kategorien zu (D'Hoinne et al., 2020). In Abbildung 3.14 sind diese Ergebnisse dargestellt.

Da es sich bei diesen Herstellern um kommerzielle Anbieter handelt, sollen nun jeweils

einer aus der jeweiligen Kategorie etwas näher beschrieben werden. *Cloudflare* als Herausforderer, *Imperva* als Führer, *Amazon Web Services* als Nischen-Spieler und *Radware* als Visionär (D'Hoinne et al., 2020).

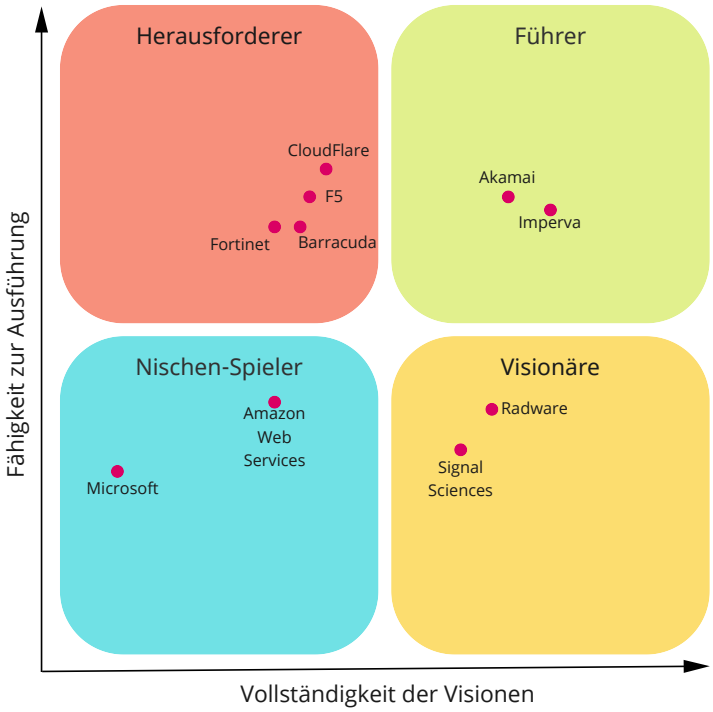


Abbildung 3.14: Gartner WAF Magic Quadrant[24]

Cloudflare

Das börsennotierte Unternehmen Cloudflare mit Sitz in San Francisco wird im Quadranten der Herausforderer gelistet und wird grundsätzlich als ernsthafter Anwärter im Bereich von Web Application und API Protection (WAAP) bewertet. Mit bereits mehr als 1500 Mitarbeitenden bietet das Unternehmen hauptsächlich Dienste im Bereich von DDoS Protection und CDN (Content Delivery Network) Netzwerken, allerdings auch eine eigene Single-Sign-On Lösung. Im Bereich von Web Application Firewalls bietet Cloudflare eine praktische Self-Service-Lösung für Unternehmen jeder Größe und mehreren Niederlassungen (D'Hoinne et al., 2020).

Cloudflare brilliert grundsätzlich mit einer ausgezeichneten Infrastruktur und bietet seine Services in einer Vielzahl von Regionen an. Darunter auch China, in welchen andere Unternehmen oftmals nicht präsent sind. Die WAF-Lösung von Cloudflare ist rein Cloud-basiert und lässt sich einfach mit den anderen Dienstleistungen des Unternehmens kombinieren (D'Hoinne et al., 2020).

[24] Grafik adaptiert von (D'Hoinne et al., 2020).

Das WAF-System hebt sich dadurch von anderen WAFs ab, dass es eines der einzigen Systeme ist, das die Verwendung von Remote-Hardware-Sicherheitsmodulen (HSM)[25] unterstützt. Darüber hinaus verfügt die WAF von Cloudflare über ein äußerst flexibles Bot-Management und ein hervorragendes Regel-Management. Für beide gibt es ein Modul, um die definierten Regeln zu testen, das zeigt wie oft eine Regel in den letzten 24 Stunden angeschlagen hat. In Kombination mit flexiblen Möglichkeiten für Reaktionsmaßnahmen wird die Benutzererfahrung für die Anwenderinnen und Anwender dadurch stark erhöht. Ein guter On-Boarding Prozess und regelmäßige Veröffentlichungen neuer Funktionen geben dem Unternehmen gute Noten von ihren Kundinnen und Kunden (D'Hoinne et al., 2020).

Allerdings ist Cloudflare in einigen Punkten noch nicht perfekt und hat Verbesserungsbedarf. Das Unternehmen agiert nach dem Prinzip schnell zu handeln und zu experimentieren. Dabei kann es schon mal zu Ausfällen kommen. Auch wenn diese meist schnell behoben werden, kann das für Unternehmen mit geringer Risikobereitschaft eine Herausforderung sein (D'Hoinne et al., 2020).

Der WAF-Lösung mangelt es darüber hinaus an einem automatisierten positiven Sicherheitsmodell und ist auch im Bereich der API-Sicherheit hinter konkurrierenden Lösungen. Zwar werden laufend fortschrittliche Funktionen hinzugefügt, allerdings hat dies eine unübersichtliche Benutzeroberfläche zur Folge. Ein weiteres Manko der Cloudflare WAF ist das Fehlen von Funktionen für grenzüberschreitende Datenübermittlung. Unternehmen haben dadurch Probleme damit, die Vorgaben der DSGVO einzuhalten, wie beispielsweise das Sicherstellen, dass der Datenverkehr nur in europäischen Rechenzentren aufgezeichnet wird. Weiters fehlt es Cloudflare an Funktionen wie Malware-Überprüfung und der Möglichkeit zur Planung von aggregierten Berichten (D'Hoinne et al., 2020).

Imperva

Das kalifornische Unternehmen Imperva wird von Gartner als Führer im Bereich von Web Application Firewalls gelistet. Die On-Premises- und Cloud-Angebote des Unternehmens bieten starken Sicherheitsschutz für deren Kundinnen und Kunden. Das Portfolio reicht von Produkten für Datensicherheit, über Runtime-Application-Self-Protection (RASP), bis hin zu Hardware-, Cloud- und virtuellen WAF Systemen. Darüber hinaus bietet Imperva Abonnements für integrierte Echtzeit-Analysen ihrer WAF-Systeme und die Möglichkeit des Betriebes der WAFs als Managed Service oder innerhalb eines Managed Security Operations Center (SOC). Das Angebot von Imperva richtet sich grundsätzlich

[25] Hardware-Komponente zur Absicherung und Verwaltung von kryptographischen Verfahren (Luber & Schmitz, 2018b).

an alle Arten von Organisationen, bietet sich aber gerade für große Unternehmen an, welche Bedarf an hochsicheren WAF-Systemen haben und verteilte WAAP-Dienste benötigen (D'Hoinne et al., 2020).

Die WAF-Systeme von Imperva besitzen umfassende Funktionalitäten für API-Sicherheit und die Produktpalette konzentriert sich grundsätzlich auf End-to-End Sicherheit von Anwendungen. Sie bieten einerseits Schutz vor DDoS-Angriffen und andererseits die Möglichkeit, komplexe serverseitige Anfragen wie JSON, XML, Websockets, GraphQL oder Remote Procedure Calls (RPC) zu analysieren. Neben den genannten Funktionalitäten bietet Imperva ebenfalls Funktionen für eine effiziente Erkennung und Abwehr von Bot-Anfragen und ermöglicht eine Simulation der definierten Regeln, um den Benutzerinnen und Benutzern zu helfen die Auswirkung derer vor der Inbetriebnahme zu testen. Durch die angebotenen Dienstleistungen hat Imperva sich eine große Basis an zufriedenen Kunden eingefahren und schneidet gerade im Bereich von WAF und WAAP wesentlich besser als konkurrierende Unternehmen ab (D'Hoinne et al., 2020).

Trotz der durchaus positiven Bewertungen der Kundinnen und Kunden und des äußerst großen Kundenstamms gibt es allerdings auch einige Punkte, die bei Imperva zu bedenken sind. Im Vergleich zur Konkurrenz und der eigenen virtuellen WAF, verfügt die Cloud-basierte WAF-Lösung beispielsweise nicht über Machine Learning zur Implementierung eines positiven Sicherheitsmodells und verlässt sich dabei auf statisch definierte Regelwerke. Die Cloud-basierte WAF-Lösung hingegen verfügt über keine Möglichkeit, um Dateien nativ auf Malware zu prüfen. Auch die geografische Präsenz des Unternehmens hält sich in Grenzen und bietet wenig Support für die Regionen China und Indien (D'Hoinne et al., 2020).

Darüber hinaus verfügt die generelle Infrastruktur von Imperva über weniger Points of Presence (POP) im Vergleich zu konkurrierenden Unternehmen. Zwar wird das Unternehmen von seinen Kundinnen und Kunden überwiegend positiv bewertet, in den Bereichen Service und Support liegt es aber hinter der Konkurrenz. Darüber hinaus scheinen einige Kundinnen und Kunden enttäuscht von der Cloud-basierten WAF-Lösung zu sein, was auf Probleme der Standardkonfigurationen und den limitierten Optionen für die Konfiguration dieser zurückzuführen ist (D'Hoinne et al., 2020).

Amazon Web Services

Amazon Web Services (AWS) wird von Gartner in den Quadranten der Nischen-Spieler gesteckt. Die Tochtergesellschaft von Amazon bietet als Cloud Service Provider neben Identitäts- und Zugriffsmanagement (IAM) ebenfalls Dienste für verwaltete Bedrohungs-

erkennung, Firewall Management und eben eine Web Application Firewall an, die AWS WAF. Das WAF-System von AWS bezieht sich grundsätzlich auf Cloud-basierte Webanwendungen, die innerhalb oder außerhalb von AWS gehostet werden und bietet grundlegende Sicherheit und Schutz vor DDoS-Attacken (D'Hoinne et al., 2020).

Dieses System bietet sich demnach gerade für bestehende Kundinnen und Kunden von AWS-Services an, die nach einer einfachen Möglichkeit suchen, ihren Applikationen Sicherheits-Signaturen voranzustellen. AWS bietet dabei einerseits Regelsätze, die von AWS selbst verwaltet werden, sogenannte *Managed Rules*, als auch Regelsätze von Drittanbietern. Zwar ist AWS im Vergleich zur Konkurrenz im Bereich von Cloud-WAFs bei einigen Funktionen im Rückstand, dennoch hat das WAF-System von AWS seine Vorteile (D'Hoinne et al., 2020).

Der Hauptvorteil der AWS WAF liegt darin, dass sie sich unkompliziert und kostengünstig mit anderen AWS-Services kombinieren lässt und dadurch für bestehende Kundinnen und Kunden eine verbesserte Erfahrung mit sich bringt. Die bemerkenswerte Cloud-Infrastruktur von AWS bietet grundsätzlich eine hohe Verfügbarkeit der vorhandenen Services und auch einige andere WAF Anbieter stellen ihre Services als virtuelle Instanzen über AWS zur Verfügung. Sowohl die AWS Managed Rules, als auch die von bekannten MSSPs bereitgestellten Regelsätze aus dem AWS Marketplace, bieten den Kundinnen und Kunden einen hohen Sicherheitsstandard und erhöhte Flexibilität. Der AWS WAF Service ist dabei vollständig programmierbar und ermöglicht den Administratoren die automatische Erstellung neuer Regeln auf Basis von beobachteten Aktivitäten wie beispielsweise SQL-Injection oder XSS (D'Hoinne et al., 2020).

Die AWS WAF zeigt allerdings auch einige Schwächen auf. Beispielsweise sind Defizite im Bereich von anwendungsspezifischen Schutzfunktionen gegenüber Bots erkennbar, welche bei konkurrierenden Produkten Standard sind. Darüber hinaus ist die AWS WAF stark auf das AWS Amazon API Gateway angewiesen, um erweiterte Verwaltungsfunktionen ausführen zu können und kann selbst keine Sicherheitsfunktionen im Bereich Mobile-Security durchsetzen. Contend Delivery Networks können nur integriert werden, wenn sie sich unmittelbar vor oder innerhalb von AWS befinden. Zusätzlich zeigt die Erfahrung von Kundinnen und Kunden, dass es zu Problemen mit False-Positive Rates und der Erstellung von Berichten kommt (D'Hoinne et al., 2020).

Radware

Radware wird aufgrund der Verfolgung eines differenzierten Ansatz als Visionär im Bereich von Web Application Firewalls eingestuft. Das Unternehmen versucht ein posi-

tives Sicherheitsmodell auf Basis von Machine-Learning für die eigene WAF-Lösung umzusetzen und setzt stark auf Innovation. Das Angebot des Unternehmens reicht vom Schutz vor DDoS-Angriffen bis hin zur Bereitstellung und Absicherung von Anwendungen. Das eigene WAF-System, AppWall, kann dabei entweder als Hardware- oder virtuelle Komponente, als auch als Teil des eigenen Cloud-Services eingesetzt werden. Die Lösungen von Radware scheinen für die meisten Unternehmen in Frage zu kommen, gerade aber für Unternehmen, die stark auf ein positives Sicherheitsmodell setzen (D'Hoinne et al., 2020).

Die Bereitschaft zur Innovation rückt Radware immer wieder in den Vordergrund, da das Unternehmen laufend in neue Architekturen und Anwendungsfälle investiert. Beispielsweise bietet das Unternehmen native Unterstützung von Kubernetes und stellt Integrationen für Kibana und Graphana bereit, um Monitoring und Reporting für DevOps-Prozesse zu ermöglichen. Die fortschrittlichen, auf Machine-Learning basierten Algorithmen können sowohl für positive als auch negative Sicherheitsmodelle eingesetzt werden und ermöglichen eine automatische Generierung von Regelwerken und Richtlinien. Darüber hinaus verfügt AppWall über einen Lern-Modus und enthält vordefinierte Vorlagen für verschiedene Anwendungstypen, wie beispielsweise PHP-Anwendungen (D'Hoinne et al., 2020).

Die Cloud-basierte WAF-Lösung basiert auf AppWall, wodurch ein bidirektionaler Übertrag der definierten Richtlinien ermöglicht wird und ein unkomplizierter Umstieg in die Cloud oder umgekehrt gewährleistet wird. Unabhängig der verwendeten Umgebung, verfügen die WAF Produkte von Radware über die gleichen Funktionalitäten und bieten somit eine konsistente Erfahrung für die Kundinnen und Kunden. Darüber hinaus verfügt auch Radware über eine vollständig integrierte Bot-Management-Lösung und softwareseitige Automatisierungen für Onboarding-Prozesse, Konfigurationsänderungen und Betriebsmanagement (D'Hoinne et al., 2020).

Wie auch vergleichbare Unternehmen, mangelt es Radware an der Präsenz in gewissen Regionen wie China oder Lateinamerika. Bestandskunden des Unternehmen kritisieren darüber hinaus lange Antwortzeiten des technischen Supports und scheinen Probleme mit der Integration zu anderen Produkten wie Security Information and Event Management (SIEM) Systemen zu haben. Radware verfügt über ein relativ kleines Bedrohungsforschungsteam im Vergleich zur Konkurrenz und verlässt sich dabei auf eigene Signaturen und die Ergebnisse des eigenen ERT-Dienstes, um die WAF-Lösungen für aktuelle Bedrohungen zu aktualisieren (D'Hoinne et al., 2020).

Auf technischer Ebene hat Radware ebenfalls noch Verbesserungsbedarf. Die Cloud-

Lösung unterstützt beispielsweise keine Anfragen, die von Servern mit IPv6 Adressen gesendet wurden. Darüber hinaus fehlt es Radware an gewissen Sicherheitsfunktionen, wie beispielsweise das Unterstützen von OAuth, API-Schlüsselverwaltung und der Analyse von Webhooks und serverseitigen Anfragen (D'Hoinne et al., 2020).

3.4.2 Open Source

Neben den gelisteten und beschriebenen Anbietern aus dem kommerziellen Bereich, gibt es auch einige Open-Source Lösungen von Web Application Firewalls. In diesem Abschnitt soll ein Einblick in solche gegeben werden. Die Auswahl erfolgt auf Basis einer Publikation von Sampaio und Bernardino (2017), welche folgende Open-Source WAF Anbieter etwas genauer unter die Lupe nehmen (Sampaio & Bernardino, 2017):

- ModSecurity
- WebCastellum
- IronBee

ModSecurity

ModSecurity ist eine der bekanntesten Open-Source Web Application Firewalls und basiert auf dem Apache HTTP Web Server und wurde unter der Apache 2.0 Lizenz veröffentlicht. Wie viele andere Open-Source Projekte auch, wurde ModSecurity im Rahmen eines Hobbys im Jahr 2002 von Ivan Ristić ins Leben gerufen. Diesem wurde bewusst, dass es unmöglich scheint, eine in jeder Hinsicht sichere Web Applikation zu entwickeln, wodurch sich die Fantasie entwickelte, ein Tool zu besitzen, dass es erlaubt, den Datenfluss einer Webapplikation zu kontrollieren noch bevor dieser die Anwendung erreicht. Die erste Version von ModSecurity wurde im gleichen Jahr veröffentlicht, bereits ein paar Monat später wurden die Funktionen von ModSecurity erst wirklich nützlich und die Popularität nahm rasant zu (Folini & Ristic, 2017).

Bereits im Jahr 2006 wurde ModSecurity als Konkurrent zu herkömmlichen WAF-Systemen gelistet und erzielte außergewöhnlich gute Ergebnisse. Noch im gleichen Jahr wurde ModSecurity von Breach Security übernommen und weiterentwickelt. 2009 entschließ Ivan Ristić, das Projekt und Breach Security zu verlassen, ein Jahr danach wurde Breach Security mitsamt ModSecurity vom Unternehmen Trustwave übernommen, welche es bis heute weiter entwickeln (Folini & Ristic, 2017). Trustwave gab allerdings

bekannt, dass der Support für ModSecurity mit erstem Juli 2024 eingestellt wird und das Projekt wieder in die Hände der Open-Source-Community übergeben wird.

ModSecurity bietet die Möglichkeit der Überwachung, Protokollierung, Fehlersuche und Zugriffskontrolle von Webanwendungen in Echtzeit und verzichtet dabei auf festgelegte Regeln und Richtlinien. Stattdessen legt es die Auswahl der Funktionen in die Hand der Anwenderinnen und Anwender und unterstreicht damit die Natur von Open-Source. Das System kann dabei entweder direkt in den verwendeten Apache HTTP-Server eingebettet oder selbstständig als Reverse-Proxy verwendet werden. Die Offenheit des Quellcodes bietet darüber hinaus die Möglichkeit der Erweiterung des Systems auf Basis der eigenen Anforderungen. Die wichtigsten Anwendungsszenarien von ModSecurity umfassen folgende (Folini & Ristic, 2017):

- Echtzeit-Zugriff auf HTTP-basierte Datenflüsse zur Überwachung und Kontrolle des Zugriffs
- Virtuelles Patching zur Behebung von Sicherheitslücken ohne Anpassungen an der Anwendung selbst
- Vollständige und flexible Protokollierung des Datenverkehrs
- Frühzeitige Erkennung von Anomalien und Sicherheitsschwächen
- Reduzierung der Angriffsfläche durch selektive Einschränkung auf Basis gewisser HTTP-Merkmale

ModSecurity kann allerdings auch Nachteile mit sich bringen. Falls die WAF nicht als Reverse-Proxy verwendet, muss ModSecurity auf jedem Web-Server installiert sein, wodurch teilweise kostbare Ressourcen verbraucht werden. Damit einhergehend ist es notwendig, Duplikate der verwendeten Konfigurationen zu erstellen, wodurch eine effiziente Wartung erschwert wird. Nachdem die verwendeten Funktionalitäten in der Hand der Anwenderinnen und Anwender liegen, müssen diese Experten im Bereich von Protokollen und Sicherheit sein, um die WAF korrekt konfigurieren zu können. Darüber hinaus müssen komplexe Regeln selbst erstellt werden, solange nicht auf eine kommerziell unterstützte Version zurückgegriffen wird. Schlussendlich wird die Wartung und die Effizienz von ModSecurity beeinträchtigt, sobald von Load-Balancing gesprochen wird (MacVittie, 2008).

WebCastellum

WebCastellum stellt einen eigenen Ansatz von Web Application Firewall dar. Diese WAF-Lösung ist eine auf Java basierte Anwendung, die sich direkt im Archiv der Anwendung selbst befindet und direkt auf diese einwirkt. Im Gegensatz zu klassischen WAF, welche nur in Bezug auf Infrastruktur funktionieren, wählt WebCastellum diesen anderen Ansatz und ist direkt mit den zu schützenden Applikationen verbunden, ohne dabei den Quellcode der Anwendung ändern zu müssen (itanius informatik GmbH, o. J.).

Auf Basis dieses Prinzips wird es ermöglicht, WebCastellum bereits während der Entwicklung zu integrieren. Der Zuschnitt der Konfiguration passiert dabei im besten Fall parallel. Allerdings ist auch eine nachträgliche Integration und Konfiguration ohne Probleme möglich. WebCastellum wird dabei als direkter Bestandteil der zu schützenden Anwendung ausgeliefert. Diese WAF hat vollen Zugriff auf die Web-Sessions als auch auf den genauen Zustand der Applikation selbst. Dadurch wird eine hochwertigere Analyse der Anwendungssicherheit gewährleistet, als herkömmliche WAFs es können (itanius informatik GmbH, o. J.).

WebCastellum basiert auf einer reinen Java-Implementierung, welche komplett ohne externe Abhängigkeiten (Dependencies) auskommt und somit die Angriffsfläche durch externe Sicherheitslücken eliminiert. Das WAF-System ist dabei mit den gängigsten JavaEE Servern kompatibel, wie beispielsweise Tomcat, BEA Weblogic, JBoss oder WebSphere und ist ein integraler Bestandteil der Applikation ohne dabei eine Änderung am eigentlichen Source-Code vornehmen zu müssen. Dabei kann die WAF nicht nur jeglichen HTTP und HTTPS Datenverkehr prüfen, sondern darüber hinaus detaillierte Protokolle für die forensische Analyse von Angriffen liefern, Web-Sitzungen im Falle eines Angriffes sofort terminieren, Regeln auf Basis von Regular-Expressions ausführen und gängige Dekodierungen vornehmen (itanius informatik GmbH, o. J.).

Weitere Funktionen von WebCastellum umfassen (itanius informatik GmbH, o. J.):

- Die flexible Administration der Regelwerke
- Definition von Ausnahmen
- Optionale Whitelist-Modelle
- Ein Lernmodus

IronBee

IronBee ist ein weiteres Open-Source Projekt einer Web Application Firewall, das vom Unternehmen Qualys verwaltet und in Zusammenarbeit mit dem Urheber von ModSecurity, Ivan Ristić, seit 2011 entwickelt wird. IronBee hat nicht nur die Entwicklung des Grundgerüsts und der Regelwerke dieser WAF zum Ziel, sondern ebenfalls eine Community rund um das Projekt aufzubauen. Auf letzteres wird dabei der Fokus gelegt, da Qualys den Erfolg von IronBee in die Hände der Community legt. Um das zu erreichen, legt Qualys viel Wert auf die Verwendung von liberalen Open-Source Lizenzen und eine Nachhaltige Gemeinschaft. Diese Schlüsselaspekte sollen dabei dafür sorgen, dass sich die Gemeinschaft durch Gleichberechtigung bilden und wachsen kann und dass es eine Vielzahl an Herausforderungen und Implementierungen gibt, die auf die unterschiedlichen Fähigkeiten und Interessen der Gemeinschaft abgestimmt sind, damit es für jeden Möglichkeiten zum Mitwirken gibt (Qualys Inc., 2011).

Das Projekt IronBee versucht dabei mit vielfältigen Einsatzmöglichkeiten die bekannten Anforderungen moderner Architekturen abzudecken und kann sowohl als passive, eingebettete, Reverse-Proxy, Command-Line und Out-Of-Process WAF eingesetzt werden. Dabei werden Abstraktionen verwendet, um den Großteil des Codes unabhängig von der verwendeten Umgebung zu machen und die Portierung von IronBee auf eine minimale Änderung auf der Schicht der Datenerfassung zu reduzieren. Modularität ist ein eben so wichtiger Aspekt bei der WAF von IronBee. Einerseits gewährleistet Modularität, dass neue Entwickler sich innerhalb des Open-Source Projektes schnell zu recht finden und nicht das gesamte System verstehen zu müssen und andererseits steigert es die Benutzererfahrung der Endanwenderinnen und Endanwender, da diese in der Lage sind, die Konfiguration der WAF auf ihre Bedürfnisse abstimmen zu können (Qualys Inc., 2011).

Für die tatsächliche Überwachung und Verteidigung der Anwendungssicherheit bietet IronBee eine Reihe von Funktionen auf unterschiedlichen Abstraktionsebenen, welche es den Benutzerinnen und Benutzern ermöglicht, einen Ansatz zu wählen, der für deren Zwecke am besten geeignet scheint (Qualys Inc., 2011):

1. Flexible Einsatz-Modi
2. Persönlichkeits-basierte Datenverarbeitung
3. ein Datenmodell, dass Entitäten wie Anwendungen, Sitzungen, Benutzer oder IP-Adressen darstellen kann

4. Aggregation historischer Daten
5. Profilierung von User-Agents
6. eine effiziente Engine zum Abruf und zur Umwandlung von Daten
7. mehrere Pattern-Matching Dienste
8. eine Auswahl an Ansätzen zur Implementierung benutzerdefinierter Sicherheitslogiken, wie eine leistungsstarke Scripting-Plattform
9. Analyse des ein- und ausgehenden Datenverkehrs
10. Sicherheitsmodule auf hoher Ebene, wie DDoS Erkennung
11. Entscheidungen auf Basis von Richtlinien
12. Maßgeschneiderte Verteidigungsmaßnahmen
13. Datenaustausch und Interaktion mit externen Sicherheitssystemen

3.4.3 Gegenüberstellung

Bei der großen Anzahl an Unternehmen, die eine Web Application Firewall anbieten, ist es nicht einfach, eine passende Wahl zu treffen. Jedes Produkt hat dabei seine eigenen Stärken, durch welche versucht wird, sich von der Konkurrenz abzuheben. Allerdings ist keines dabei perfekt oder kann einen 100 prozentigen Schutz der Anwendungen garantieren. Bei der Geschwindigkeit, mit der sich der technische Fortschritt entwickelt, und bei der immer größer werdenden Anzahl an Bedrohungen, die damit einhergeht, ist das auch nicht möglich.

Das stellt die Unternehmen, die Bedarf an einer WAF haben, vor eine Herausforderung. Um die richtige Wahl zu treffen, müssen die eigenen Systeme und Anforderungen an diese genau analysiert und verstanden werden. Die Wahl sollte dabei nicht nur für den aktuellen Moment getroffen werden, sondern darüber hinaus die mittel- bis langfristigen Pläne des Unternehmens beachten. Plant das Unternehmen beispielsweise seine Dienste großteils in die Cloud zu verlagern, sollte keine Hardware-basierte On-Premises WAF angeschafft werden.

Gleichzeitig sind natürlich die Kosten ebenfalls ein Punkt der beachtet werden muss.

Zwar wurde in dieser Arbeit auf einen Kosten-Vergleich verzichtet, in der heutigen Zeit gehören allerdings Lizenz- oder Wartungsverträge zum Standard. Gerade bei kommerziellen Anbietern können versteckte Kosten anfallen, die sich schnell aufsummieren. Dafür ist von diesen meist ein langlebiger Produkt-Support gewährleistet und einige bieten ihre WAF-Lösung als Managed Service an.

Im Open-Source Bereich sind solche Garantien oder fortlaufende Verbesserung der Dienste nicht garantiert. Zwar bieten diese eine kostenlose oder kostengünstige Alternative zu kommerziellen Anbietern, allerdings erwirbt man bei solchen Lösungen meist nur das Grundgerüst und ist selbst für Aktualisierungen und Erweiterungen verantwortlich. Ohne ein dediziertes Team kann eine Open-Source Lösung wohl nicht effektiv betrieben werden.

Allerdings hat es den Anschein, dass Open-Source WAFs innovativere Ansätze wählen, als kommerzielle Anbieter. Gleichzeitig hat dies allerdings die Konsequenz, dass die angebotenen Funktionen limitiert sind und in ihrer Anzahl wesentlich geringer ausfallen. Updates oder neue Funktionen lassen bei Systemen, die von einer Community entwickelt werden, gerne etwas auf sich warten. Im kommerziellen Bereich werden regelmäßig neue Funktionen und Verbesserungen ausgerollt und es kann wesentlich schneller auf neue Bedrohungen reagiert werden.

Ob es nun besser ist, direkt ein kommerzielles Produkt zu erwerben, oder sich auf eine Open-Source Lösung zu stützen, kann im theoretischen Kontext nicht entschieden werden. Diese Entscheidung ist abhängig von den eigenen Anforderungen und Systemen, von der Bereitschaft zur eigenen Wartung, den verfügbaren finanziellen Mitteln und dem langfristigen Plänen des Unternehmens.

4 Cookie Validierung mit ModSecurity

Bisher wurde in dieser Arbeit darauf eingegangen, warum die Absicherung von Applikationen und Systemen in der heutigen Zeit so wichtig ist und welchen bekannten Bedrohungen diese ausgesetzt sind. In diesem Kontext wurden Web Application Firewalls im Detail beschrieben und erläutert, wie deren Einsatz zum Schutz der dahinterliegenden Applikationen für ein hohes Maß an Sicherheit sorgen können. Darüber hinaus wurde ein Einblick in eine Auswahl an bestehenden WAF Lösungen sowohl im kommerziellen als auch im Open-Source Bereich gegeben und deren Vor- und Nachteile untersucht.

In folgendem Abschnitt der Arbeit wird nun etwas tiefer in eines dieser bestehenden Systeme eingetaucht und dessen Funktionsweise praktisch veranschaulicht. Es wird versucht, anhand eines realitätsnahen Szenarios, die Integration der Open-Source WAF ModSecurity zum Schutz einer Web-Applikation praxisnah darzustellen. In Kapitel 3.1.3 wurde bereits erläutert, dass Web Application Firewalls grundsätzlich die Möglichkeit bieten, Cookies digital zu signieren und in weiterer Folge deren Integrität zu validieren. Standardmäßig wird diese Funktionalität nicht von ModSecurity unterstützt, allerdings sollte sich so eine Funktionalität anhand von selbst definierten Regeln umsetzen lassen.

Dieses Kapitel widmet sich dem Konzept und der Umsetzung einer solchen Funktionalität. Zu Beginn wird ein Konzept erarbeitet, das veranschaulichen soll, wie eine Cookie-Validierung mithilfe einer Web Application Firewall aussehen könnte. Weiters werden auf Basis dieses Konzepts die Anforderungen für eine praktische Implementierung analysiert und erläutert. Schlussendlich wird versucht, dieses Konzept in die Praxis umzusetzen und mithilfe der Open-Source WAF ModSecurity Cookies zu signieren und zu validieren.

4.1 Konzept

Wie bereits erwähnt wurde, sind HTTP-Cookies ein wichtiger Bestandteil in der Kommunikation über das HTTP-Protokoll und werden oft für die Identifizierung einzelner Clients gegenüber einem Web-Server verwendet. Allerdings können diese von Angreiferinnen und Angreifern gestohlen, manipuliert oder vorhergesagt werden, um sich unrechtmäßigen Zugriff auf den Web-Server zu verschaffen. Um das zu verhindern, bedarf es einer Funktionalität, um solche eingehenden Cookies zu validieren und gegebenenfalls Anfragen zu blockieren, falls die Integrität der Cookies nicht gegeben ist.

Um ein Konzept für die Integritätsprüfung von Cookies bereitstellen zu können, ist es notwendig, die Funktionsweise von Cookies zu verstehen. Cookies sind kleine Datenpakete, die mit der Antwort auf eine Anfrage bei Bedarf vom Server an den Client gesendet und am Client gespeichert werden. Sendet der gleiche Client erneut eine Anfrage an denselben Server, werden die Cookies automatisch mitgesendet und können vom Server ausgewertet werden (Mozilla Corporation, 2022).

Abbildung 4.1: HTTP-Cookie Ablauf[1]

Cookies werden von Web-Servern über einen Set-Cookie HTTP-Header gesetzt und bestehen aus einer eindeutigen Bezeichnung und dem entsprechenden Wert. Darüber hinaus können Cookies mit verschiedenen Attributen versehen werden, wie beispielsweise einem Ablaufdatum oder einem HTTP-Only Kennzeichen, welcher den Zugriff auf den Cookie innerhalb von JavaScript untersagt (Mozilla Corporation, 2022). Der grobe Aufbau eines Set-Cookie Header ist in Listing 4.1 dargestellt.

```
Set-Cookie: <cookie-name>=<cookie-value>; Expires=<date>; HttpOnly
```

Listing 4.1: Cookie Aufbau [2]

[1] Grafik vom Autor erstellt.
[2] Adaptiert von (Mozilla Corporation, 2022).

Cookies beinhalten ihre Informationen meist in unverschlüsselter Form und können daher direkt ausgelesen werden. Das ermöglicht Angreiferinnen und Angreifern sie für ihre Zwecke zu missbrauchen. Daher sollten Cookies entweder verschlüsselt oder dahingehend validiert werden, dass sichergestellt werden kann, dass diese nicht verändert wurden. Das kann grundsätzlich leicht von der Web-Applikation selbst gemacht werden. Allerdings gibt es Szenarien, in welchen man keinen Einfluss auf die Applikation an sich hat, wodurch sich eine solche Funktionalität durch den Einsatz einer WAF ebenfalls anbieten würde. Beispielsweise könnte die WAF aus einem ausgehenden Set-Cookie Header eine Signatur berechnen und diese als weiteren Cookie übermitteln. Kommen die Cookies mit der nächsten Anfrage wieder retour, kann die Signatur erneut berechnet und verglichen werden, um die Integrität des Cookies zu prüfen.

Diese Überlegung soll als Basis für das Konzept der praktischen Umsetzung dienen und ist in Abbildung 4.2 grafisch dargestellt.

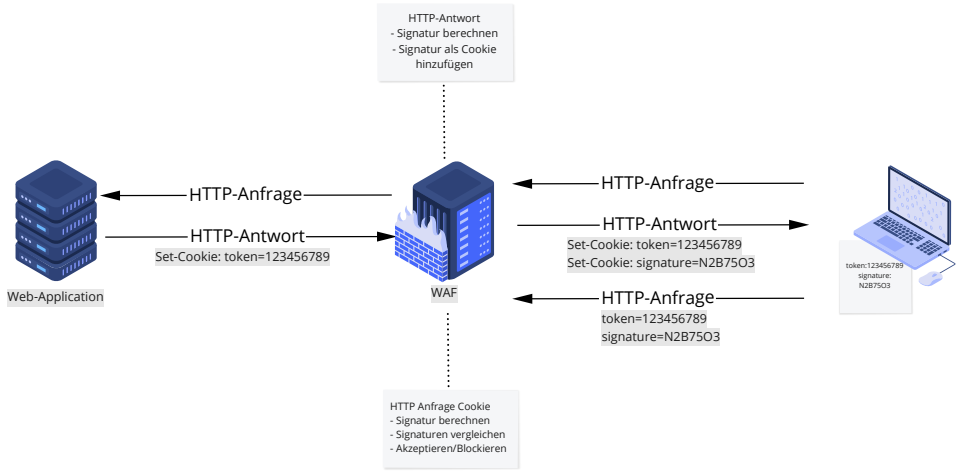


Abbildung 4.2: Konzept Cookie Signatur[3]

In der Abbildung wird von einem Client eine Anfrage an eine Web-Applikation gesendet. Dieser setzt in seiner Antwort einen Token mithilfe eines Set-Cookie HTTP-Headers. Die dazwischenliegende Web Application Firewall erkennt den ausgehenden Set-Cookie Header und berechnet eine Signatur auf Basis der Informationen des Cookies. Diese Signatur wird ebenfalls mithilfe eines Set-Cookie Header an den Client übermittelt.

Sendet der Client nun erneut eine Anfrage an die Web-Applikation, übermittelt er die gesetzten Cookies innerhalb der neuen Anfrage. Die WAF erkennt, dass beide Cookies vorhanden sind und kalkuliert auf Basis des initialen Cookies erneut die Signatur. Daraufhin wird die berechnete Signatur mit der als Cookie übermittelten Signatur verglichen.

[3] Grafik vom Autor erstellt.

Stimmen die Signaturen überein, wird die Anfrage an die Web-Applikation weitergeleitet. Stimmen die Signaturen nicht überein, wird die Anfrage des Clients blockiert.

Um dieses Konzept in die Praxis umzusetzen, wird im folgenden Kapitel ein kleines Test-Szenario aufgebaut. Anhand von diesem Test-Szenario wird in weiterer Folge ein eigenes Regelwerk in ModSecurity erstellt werden, um die beschriebenen Anforderungen in die Praxis umzusetzen.

4.2 Test-Szenario

Um den großen Aufwand eines Server-ähnlichen Test-Systems auf einem Minimum zu halten, wird das Test-Szenario auf einem einzelnen virtualisierten Debian Linux Server aufgesetzt. Die verwendeten Komponenten können allerdings alle betriebssystemunabhängig verwendet werden. Für das Setup des Test-Szenarios orientieren wir uns auf dem im vorhinein erstellten Konzept. Demnach werden für das Test-Szenario die folgenden Komponenten benötigt:

- Eine Web-Applikation zum Platzieren von Cookies.
- Ein Apache Web-Server, welcher als Proxy fungiert.
- Eine Installation von ModSecurity am Apache Web-Server.

4.2.1 Web-Applikation

Einerseits wird für das Test-Szenario eine Web-Applikation benötigt, die demonstrativ einen Cookie auf Clients platziert, die eine Anfrage senden. Um dies zu simulieren wurde eine minimale Web-Applikation in Java entwickelt, die für Testzwecke einen Endpunkt für eingehende Anfragen bereitstellt und lediglich einen Cookie an den Client übermittelt. Diese Micro-Applikation verfügt über folgenden Rest Controller:

```
import org.springframework.http.HttpEntity;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;

@RestController
public class CookieController {

    @GetMapping(path = "/cookieJar")
    public HttpEntity<String> cookieJar(final HttpServletResponse response) {
        Cookie cookie = new Cookie("my-delicious-cookie", "caramel");
        response.addCookie(cookie);
        return ResponseEntity.ok("Enjoy your cookie!");
    }
}
```

Listing 4.2: Cookie Controller

Der in Listing 4.2 abgebildete CookieController macht nichts anderes, als einen REST-Endpunkt mit dem Pfad */cookieJar* für einen HTTP-GET-Request bereit zu stellen. Wird dieser Endpunkt nun von einer Anfrage erreicht, setzt die Web-Applikation einen Cookie mit dem Namen *my-delicious-cookie* und dem Wert *caramel* für den Client und antwortet mit der Meldung *Enjoy your cookie!*.

Das Ergebnis einer Anfrage an diesen Endpunkt lässt sich einfach mit dem Kommandozeilenbefehl

curl -v localhost:9090/cookieJar

überprüfen und ist in Listing 4.3 dargestellt. Es zeigt den Set-Cookie Header, *Set-Cookie: my-delicious-cookie=caramel*, der von der Applikation gesetzt wird und ebenfalls den retournierten Body *Enjoy your cookie!*.

```
*   Trying 127.0.0.1:9090...
* Connected to localhost (127.0.0.1) port 9090 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost:9090
> User-Agent: curl/7.83.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Set-Cookie: my-delicious-cookie=caramel
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 18
< Date: Sun, 29 May 2022 14:23:26 GMT
<
* Connection #0 to host localhost left intact
Enjoy your cookie!%
```

Listing 4.3: Curl GET Anfrage

4.2.2 ModSecurity Modul

Als weitere Komponente für das Test-Szenario fehlt noch die Installation von ModSecurity als Modul des Apache Web-Servers. Dies kann grundsätzlich über zwei Arten erfolgen. Einerseits über einen Package Manager, wie *apt*, andererseits wiederum kann der Source Code direkt vom entsprechenden Github Repository heruntergeladen werden und das Modul direkt vom Quellcode installiert werden. Im Rahmen dieser Arbeit beschränken wir uns allerdings auf das vorkompilierte Modul und können dies wieder bequem mit folgendem Befehl herunterladen (Folini & Ristic, 2017):

sudo apt install libapache2-mod-security2

Nach erfolgreichem Download des Moduls muss dieses wiederum aktiviert werden. Dies erfolgt über den folgenden Befehl (Folini & Ristic, 2017):

sudo a2enmod security2

Durch die Installation des Moduls wird ein neuer Ordner mit wichtigen Konfigurations-Dateien unter */etc/modsecurity* angelegt. Da ModSecurity standardmäßig nicht aktiviert

ist, wird die existierende Konfigurations-Datei mit folgendem Befehl aktiviert (Folini & Ristic, 2017):

cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf

Dieser Befehl kopiert die Standardkonfiguration und aktiviert sie gleichzeitig durch Entfernen des Suffix *-recommended*. Innerhalb dieser Konfiguration muss außerdem noch die Betriebsart von ModSecurity geändert werden, da diese standardmäßig auf *DetectionOnly* gesetzt ist. Um dies zu tun, muss die ModSecurity Rules Engine wie in Listing 4.4 dargestellt auf *On* gestellt werden (Folini & Ristic, 2017).

```
# -- Rule engine initialization ↩
   ----------------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-↩
   installation
# disruption.
#
SecRuleEngine On
```

Listing 4.4: ModSecurity Rules Engine Aktivierung

Nachdem die Aktivierung vorgenommen wurde, benötigt der Apache Server wiederum einen Neustart, damit die Änderungen tatsächlich aktiviert werden.

Danach kann die Funktionsweise von ModSecurity anhand einer einfachen Regel überprüft werden. Dazu wird exemplarisch eine Regel in der Konfigurations-Datei des Virtual Host erstellt, welche jegliche Anfrage an den Pfad */cookieJar* blockieren soll. Die entsprechende Regel sieht folgendermaßen aus (Folini & Ristic, 2017):

```
SecRule REQUEST_URI "/cookieJar" "id:327,deny,status:403,msg:'Block ↩
   Cookie Request!'"
```

Listing 4.5: ModSecurity Beispiel Regel

Hierbei indiziert das Stichwort *SecRule* eine neue Regel. Danach wird die Variable definiert, auf welche zugegriffen werden soll. In diesem Fall der URI der Anfrage. Darauf folgt ein Operator, mit welchem ein regulärer Ausdruck, ein Pattern oder ein Schlüsselwort angegeben wird, nach welchem in der Variable gesucht werden soll. Schlussendlich folgen noch Aktionen, die ausgeführt werden, nachdem eine Anfrage die Kriterien der Regel erfüllt (Folini & Ristic, 2017). Im Beispiel soll die Anfrage abgelehnt werden und mit dem

HTTP-Status Code 403 antworten. Darüber hinaus ist eine Nachricht mit den Worten *Block Cookie Request!* definiert, welche bei einem positiven Treffer in die Log-Datei geschrieben wird.

Startet man den Apache Server neu und testet die neu definierte Regel wieder mithilfe von Curl, ist die Auswirkung der Regel gut ersichtlich:

```
$ curl -v localhost:80/cookieJar
*   Trying ::1:80...
* Connected to localhost (::1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 403 Forbidden
< Date: Fri, 03 Jun 2022 16:58:43 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Length: 274
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.53 (Debian) Server at localhost Port 80</address>
</body></html>
* Connection #0 to host localhost left intact
```

Listing 4.6: ModSecurity Regel Match

Der Server antwortet in diesem Fall mit dem konfigurierten Status Code 403 und einer entsprechenden Meldung, dass der Zugriff auf die angefragte Resource verweigert wurde. Auch in der entsprechenden Log-Datei */var/log/apache2/modsec_audit.log* ist die konfigurierte Fehlermeldung zu finden (Folini & Ristic, 2017):

```
Message: Access denied with code 403 (phase 2). Pattern match "/↩
    cookieJar" at REQUEST_URI. [file
"/etc/apache2/sites-enabled/000-default.conf"] [line "44"] [id "327"] [↩
    msg "Block Cookie Request!"]
Apache-Error: [file "apache2_util.c"] [line 271] [level 3] [client ::1] ↩
    ModSecurity: Access denied with code 403
(phase 2). Pattern match "/cookieJar" at REQUEST_URI. [file "/etc/↩
    apache2/sites-enabled/000-default.conf"] [line
"44"] [id "327"] [msg "Block Cookie Request!"] [hostname "localhost"] [↩
    uri "/cookieJar"] [unique_id
"Ypo9w64eTQfKgGY5cpxZtQAAAAY"]
Action: Intercepted (phase 2)
Apache-Handler: proxy-server
Stopwatch: 1654275523231622 2308 (- - -)
Stopwatch2: 1654275523231622 2308; combined=1394, p1=575, p2=615, p3=0, ↩
    p4=0, p5=204, sr=38, sw=0, l=0, gc=0
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.3 (http://www.modsecurity.org/); ↩
    OWASP_CRS/3.3.0.
Server: Apache/2.4.53 (Debian)
Engine-Mode: "ENABLED"
```

Listing 4.7: ModSecurity Audit Log

Somit wurde sichergestellt, dass ModSecurity korrekt installiert und aktiviert wurde.

4.3 Praktische Umsetzung

Da nun ein Test-Szenario aufgesetzt und initial getestet wurde, wird nun versucht, das zuvor beschriebene Konzept mithilfe von ModSecurity umzusetzen. Die konfigurierte Rules-Engine von ModSecurity wird dabei in der Lage sein, den gesetzten Cookie der Web-Applikation zu erkennen, eine Signatur zu berechnen und als weiteren Cookie an den Client zu übermitteln. Beim nächsten Request des Clients wird die WAF die Signatur auf Basis des initialen Cookies erneut berechnen und mithilfe der mitgelieferten Signatur validieren.

Die Regeln von ModSecurity können in unterschiedlichen Phasen angewendet werden. In der ersten Phase hat man lediglich Zugriff auf die Header der Anfrage. In der zweiten Phase hat man Zugriff auf den Body der Anfrage. Die Phasen drei und vier ermöglichen den Zugriff wiederum auf die Header der Server-Antwort und separat auf den Body der

Server-Antwort. Die fünfte und letzte Phase ist rein für das Logging zuständig (Folini & Ristic, 2017).

Um initial erkennen zu können, dass die Web-Applikation überhaupt einen Cookie gesetzt hat, muss demnach in der dritten Phase auf die Header der Server-Antwort zugegriffen werden. Dies kann über folgende ModSecurity Regel erreicht werden:

```
SecRule RESPONSE_HEADERS:Set-Cookie "my-delicious-cookie" "id:9990,phase:3,deny,status:403,log,msg:'Detected Cookie my-delicious-cookie'"
```

Listing 4.8: ModSecurity Cookie erkennen

Eine Regel wird in ModSecurity grundsätzlich mit dem Schlüsselwort *SecRule* eingeleitet. Danach folgt die Deklaration der Variable auf welche zugegriffen werden soll. In diesem Fall wird auf die Header der Server-Antwort zugegriffen werden, speziell auf die Set-Cookie Header. Danach folgt eine Regular Expression, nach welcher die davor definierte Variable validiert wird. Im Beispiel wird geprüft, ob im Set-Cookie Header der Wert *my-delicious-cookie* vorkommt. Schlussendlich folgen noch gewisse Aktionen, die bei einem positiven Match ausgeführt werden sollen. Einerseits wird für die Regel eine eindeutige ID mit dem Wert *9990* festgelegt und definiert, dass sie in Phase drei anzuwenden ist, also dann, wenn die Header der Server-Antwort verfügbar sind. Um zu erkennen, ob der Cookie auch tatsächlich von der Web-Applikation gesetzt wird, wird die Anfrage exemplarisch mit dem Schlüsselwort *deny* blockiert und der Server soll mit dem Status *403* antworten. Darüber hinaus wird noch eine entsprechende Nachricht in die Log-Datei geschrieben: *Detected Cookie my-delicious-cookie!*.

Nach einem Test der Konfiguration und einem Neustart des Servers kann die erstellte Regel getestet werden. Eine erneute Anfrage über Curl zeigt in Listing 4.9, dass die Anfrage mit dem Status 403 blockiert wird.

```
curl -v localhost:80/cookieJar
*   Trying ::1:80...
* Connected to localhost (::1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 403 Forbidden
< Date: Mon, 06 Jun 2022 15:44:27 GMT
< Server: Apache/2.4.53 (Debian)
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.53 (Debian) Server at localhost Port 80</address>
</body></html>
* Connection #0 to host localhost left intact
```

Listing 4.9: ModSecurity Cookie Blockieren

Ebenfalls im Error-Log vom Apache Server, in Listing 4.10, ist die konfigurierte Fehlermeldung zu sehen. Demnach ist die WAF in der Lage, den gesetzten Cookie der Web-Applikation zu erkennen.

```
[Mon Jun 06 17:46:59.595650 2022] [:error] [pid 4668:tid ↩
    140042385794816] [client ::1:33822] [client ::1] ModSecurity: Access ↩
    denied with code 403 (phase 3). Pattern match "my-delicious-cookie" ↩
    at RESPONSE_HEADERS:Set-Cookie. [file "/etc/apache2/sites-enabled↩
    /000-default.conf"] [line "65"] [id "9990"] [msg "Detected Cookie my-↩
    delicious-cookie"] [hostname "localhost"] [uri "/cookieJar"] [↩
    unique_id "Yp4hcwhUdZD25RUxkAA-GQAAAAI"]
```

Listing 4.10: ModSecurity Cookie Blockieren Log

Im nächsten Schritt muss nun der Wert des Set-Cookie Header aus den Headern der Server-Antwort extrahiert werden. Dazu ändern wir die erstellte Regel und verketten

sie mit dem Schlüsselwort *chain* mit weiteren Regeln, welche den Wert aus dem Set-Cookie Header extrahieren und in die *TX Sammlung (TX Collection)* speichern, um in weiterer Folge darauf zugreifen zu können (Folini & Ristic, 2017). In Listing 4.11 sind die entsprechenden Regeln definiert.

```
SecRule RESPONSE_HEADERS:Set-Cookie "my-delicious-cookie" "id:9990,phase↩
    :3,chain,deny,log,msg:'Detected
Server Cookie with value %{MATCHED_VAR}'"
SecRule RESPONSE_HEADERS:Set-Cookie "my-delicious-cookie=(.*)" "capture,↩
    chain"
SecRule TX:1 "\w+"
```

Listing 4.11: ModSecurity Cookie Wert Extrahieren

Die erste Regel erkennt wiederum nur, ob der Cookie von der Applikation gesetzt wurde. Die zweite Regel extrahiert den Wert des Cookies mit dem Namen *my-delicious-cookie* anhand eines regulären Ausdruckes und speichert diesen mithilfe des Schlüsselwortes *capture* in die TX Collection. In der dritten Regel wird geprüft, ob sich in der ersten Gruppe der TX Collection ein Wert befindet. Falls ja, wird die Anfrage exemplarisch geblockt. Wie wir in der Antwort einer weiteren Curl Test-Anfrage erkennen können, wird die Anfrage blockiert und auch der Wert des Cookies wird entsprechend in der Log-Datei in Listing 4.12 ausgegeben:

```
[Mon Jun 06 18:39:21.553351 2022] [:error] [pid 5101:tid ↩
    140042391746304] [client ::1:33838] [client ::1]
ModSecurity: Access denied with code 403 (phase 3). Pattern match "\\\\w↩
    +" at TX:1. [file
"/etc/apache2/sites-enabled/000-default.conf"] [line "67"] [id "9990"] [↩
    msg "Detected Server Cookie with value
caramel"] [hostname "localhost"] [uri "/cookieJar"] [unique_id "↩
    Yp4tuT0l0Y0nkHMppXyTPgAAAAA"]
```

Listing 4.12: ModSecurity Cookie Wert Extrahieren und blockieren

Da wir nun den Wert des gesetzten Cookies in der TX Collection gespeichert haben, muss in weiterer Folge noch ein Hash-Wert berechnet werden, welcher wiederum als eigenständiger Cookie an den Client übergeben wird. Um dies zu tun, modifizieren wir die in Listing 4.11 angeführte dritte Regel und verwenden zur Veranschaulichung zwei in ModSecurity integrierte Transformations-Funktionen:

```
SecRule TX:1 "\w+" "t:none,t:md5,t:hexEncode,setenv:cookie_signature=%{↩
    MATCHED_VAR}"
```

Listing 4.13: ModSecurity Hash berechnen

Die modifizierte Regel in Listing 4.13 prüft wiederum, ob die TX Collection am Index eins einen Wert beinhaltet und führt im positiven Falle die Aktionen beziehungsweise Transformationen *t:none*, *t:md5* und *t:hexEncode* auf. Die erste Transformation sorgt dafür, dass keine davor definierten Standard-Transformationen zur Anwendung kommen. Die zweite berechnet aus dem Wert von TX:1 einen MD5-Hash-Wert im Binärformat. Mithilfe der dritten Transformation wird das Binärformat in ein Hex-Format konvertiert (Folini & Ristic, 2017). Das Ergebnis dieser Transformationen, also ein Hex-kodierter String Wert, wird in der letzten Anweisung mithilfe von *setenv* als Umgebungsvariable gespeichert.

Mithilfe des Apache Modules *mod_headers*, das zuvor mit dem Befehl

sudo a2enmod headers

geladen werden muss, kann die definierte Umgebungsvariable nun als weiterer Set-Cookie Header an den Client übermittelt werden. Die Anweisung an den Apache Server ist in Listing 4.14 dargestellt.

```
Header add Set-Cookie "signature=%{COOKIE_SIGNATURE}e" env=↩
    cookie_signature
```

Listing 4.14: ModSecurity Hash als Cookie übergeben

Diese Anweisung fügt der Anfrage einen zusätzlichen Cookie mit dem Namen *signatur* und dem Wert der Umgebungsvariable hinzu. Ein erneuter Test über Curl zeigt das Ergebnis in Listing 4.15. Der initial gesetzte Cookie und die neu erstellte Signatur sind jeweils als Set-Cookie Header gesetzt.

```
curl -v localhost:80/cookieJar
*   Trying ::1:80...
* Connected to localhost (::1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Date: Tue, 07 Jun 2022 19:57:56 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 18
< Set-Cookie: my-delicious-cookie=caramel
< Set-Cookie: signature=8661283628ec0a7b0957e9c44cf77db5
<
* Connection #0 to host localhost left intact
Enjoy your cookie!
```

Listing 4.15: ModSecurity Cookie Hash Ergebnis

Somit wurde die notwendige Funktionalität, die von der Web-Applikation in Richtung Client benötigt wird, in ModSecurity integriert und es fehlt lediglich die Validierung der gesetzten Cookies, sobald der Client erneut eine Anfrage sendet. Dazu müssen wir einerseits erkennen, dass mit dem initialen Cookie *my-delicious-cookie* ebenfalls die Signatur übermittelt wird. Ist dies nicht der Fall, soll die Anfrage blockiert werden. Diese Logik kann mit folgenden Regeln erreicht werden:

```
SecRule REQUEST_HEADERS:Cookie "my-delicious-cookie" "id:9991,log,msg:'↩
    Block request without cookie signature',chain,
deny"
SecRule REQUEST_HEADERS:Cookie "!signature"
```

Listing 4.16: ModSecurity Cookie Übermittlung prüfen

In der ersten Regel wird geprüft, ob in den Headers der Anfrage der initiale Cookie *my-delicious-cookie* gesetzt ist. In einer zweiten verketteten Regel wird darüber hinaus geprüft, ob der Wert *signatur* nicht in den Headers vorkommt. Ist dies der Fall, wird die Anfrage blockiert. Übergibt man nun mithilfe von Curl nur den Ersten der beiden Cookies, sieht man, dass die Anfrage blockiert wird:

```
curl -v --cookie "my-delicious-cookie=caramel" localhost:80/cookieJar
*   Trying ::1:80...
* Connected to localhost (::1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost
> User-Agent: curl/7.74.0
> Accept: */*
> Cookie: my-delicious-cookie=caramel
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 403 Forbidden
< Date: Tue, 07 Jun 2022 20:21:27 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Length: 274
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.53 (Debian) Server at localhost Port 80</address>
</body></html>
* Connection #0 to host localhost left intact
```

Listing 4.17: ModSecurity Cookie Übermittlung blockieren

Ergänzt man allerdings noch die Signatur, wird die Anfrage akzeptiert:

```
curl -v --cookie "my-delicious-cookie=caramel" --cookie "signature=123↩
    jkfd" localhost:80/cookieJar
*   Trying ::1:80...
* Connected to localhost (::1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost
> User-Agent: curl/7.74.0
> Accept: */*
> Cookie: signature=123jkfd
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Date: Tue, 07 Jun 2022 20:22:17 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 18
< Set-Cookie: my-delicious-cookie=caramel
< Set-Cookie: signature=8661283628ec0a7b0957e9c44cf77db5
<
* Connection #0 to host localhost left intact
Enjoy your cookie!
```

Listing 4.18: ModSecurity Cookie Übeermittlung zulassen

Um die Integrität des initialen Cookies zu validieren, muss in weiterer Folge nochmals erneut die Signatur berechnet werden, um diese mit der vom Client übermittelten Signatur vergleichen zu können. Ähnlich wie bereits zuvor, erstellen wir dazu zuerst eine Regel, die die Cookies in der Anfrage erkennt und direkt in die TX Collection speichert. In einer zweiten verketteten Regel wird daraufhin wieder die Signatur nach dem gleichen Prinzip wie zuvor berechnet: Zuerst wird der MD5-Hash berechnet und das Ergebnis daraufhin als Hex-Wert kodiert und in einer weiteren Variable gespeichert. In einer dritten verketteten Regel wird somit nur noch überprüft, ob die übermittelte Signatur der neu berechneten entspricht. Falls nicht, wird die Anfrage blockiert. Die dazu erstellten Regeln sind in Listing 4.19 dargestellt.

```
SecRule REQUEST_HEADERS:Cookie "my-delicious-cookie=(.*);.*signature↩
    =(.*)" "id:9992,phase:1,capture,chain,deny,log,
msg:'Cookie Integrity Violation: Calculated: %{tx.new-signature}, ↩
    Received: %{tx.2}'"
SecRule TX:1 "\w+" "t:none,t:md5,t:hexEncode,setvar:tx.new-signature=%{↩
    MATCHED_VAR},chain"
SecRule TX:NEW-SIGNATURE "!eq
```

Listing 4.19: ModSecurity Regel: Signatur Vergleich

In der ersten Regel werden mit dem Stichwort *capture* die beiden durch Klammern dargestellten Ausdrücke der Regular Expression in die TX Collection aufgenommen. In der zweiten Regel wird der erste der beiden Ausdrücke, demnach der Wert des normalen Cookies, in eine neue Signatur nach dem gleichen Prinzip transformiert und in einer neuen Variable gespeichert. Schlussendlich wird diese neue Signatur mit der übermittelten aus der Variable TX.2 anhand des negierten *@eq* Operators verglichen. Stimmen die Werte nicht überein, so wird die Anfrage blockiert.

Das Ergebnis lässt sich wiederum gut mit Curl testen:

```
curl -j -v --cookie "my-delicious-cookie=caramel" --cookie "signature↩
    =8661283628ec0a7b0957e9c44cf77db5"
localhost:80/cookieJar
*   Trying 127.0.0.1:80...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost:80
> User-Agent: curl/7.83.0
> Accept: */*
> Cookie: my-delicious-cookie=caramel;signature=8661283628↩
    ec0a7b0957e9c44cf77db5
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Date: Thu, 09 Jun 2022 20:48:03 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 18
< Set-Cookie: my-delicious-cookie=caramel
< Set-Cookie: signature=8661283628ec0a7b0957e9c44cf77db5
<
* Connection #0 to host localhost left intact
Enjoy your cookie!
```

Listing 4.20: ModSecurity Signatur Übereinstimmung

```
curl  -v --cookie "my-delicious-cookie=caramel" --cookie "signature=123"↩
    localhost:80/cookieJar
*   Trying 127.0.0.1:80\ldots
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cookieJar HTTP/1.1
> Host: localhost:80
> User-Agent: curl/7.83.0
> Accept: */*
> Cookie: my-delicious-cookie=caramel;signature=123
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 403 Forbidden
< Date: Thu, 09 Jun 2022 20:46:30 GMT
< Server: Apache/2.4.53 (Debian)
< Content-Length: 276
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.53 (Debian) Server at localhost Port 80</address>
</body></html>
* Connection #0 to host localhost left intact
```

Listing 4.21: ModSecurity Signatur Fehler

In der zweiten Anfrage aus Listing 4.21 wurde exemplarisch der Wert der übermittelten Signatur geändert. Es ist gut zu erkennen, dass die Anfrage erfolgreich blockiert wurde. Darüber hinaus ist in der Log-Datei die in der Regel definierte Fehlermeldung ersichtlich:

```
[Thu Jun 09 22:46:30.614011 2022] [:error] [pid 9013:tid ↩
   140042277103360] [client 10.0.2.2:60009] [client 10.0.2.2] ↩
   ModSecurity: Access denied with code 403 (phase 1). Match of "eq %{TX↩
   .2}" against "TX:new-signature" required. [file "/etc/apache2/sites-↩
   enabled/000-default.conf"] [line "67"] [id "9992"] [msg "Cookie ↩
   Integrity Violation: Calculated: 8661283628ec0a7b0957e9c44cf77db5, ↩
   Received: 123 "] [data "localhost:80"] [hostname "localhost"] [uri "/↩
   cookieJar"] [unique_id "YqJcJgwfsU49Z7Uc7j3DkwAAAEA"]
```

Listing 4.22: ModSecurity Signatur Fehler

Das entwickelte Konzept wurde somit erfolgreich umgesetzt und die Web-Applikation braucht sich keine Sorgen mehr um die Integrität der Cookies zu machen. Anhand dieses praktischen Beispiels wurde veranschaulicht, dass auch komplexere Anforderungen, welche standardmäßig kein Bestandteil von ModSecurity sind, relativ einfach mit dem bereitgestellten Regelwerk umgesetzt werden können.

Die Syntax des Regelwerks und auch die Dokumentation von ModSecurity scheinen für Laien gerade zu Beginn recht verwirrend zu sein, mit etwas Übung findet man sich in der Welt von ModSecurity aber schnell zu recht. Die bereits integrierten Funktionalitäten bieten eine vielseitige Basis für jegliche Anforderungen. In Kombination mit einer umfangreichen Regular Expression Engine, Folgeaktionen und einer nahtlosen Integration in den Apache2 Web Server, bietet ModSecurity ein mächtiges Tool, um die Kontrolle über den ein- und ausgehenden HTTP-Traffic zu behalten. Bekannte Bedrohungen können dabei bereits mit öffentlich verfügbaren Regelwerken abgedeckt werden und neue Anforderungen durch eigene Regeln oder auch Erweiterungen an ModSecurity selbst.

5 Kritische Reflexion

Das heikle Thema IT-Sicherheit rückt in der heutigen Zeit immer mehr in den Vordergrund. Unternehmen stehen vor der Herausforderung, die eigenen Systeme, das Know-How und die Daten ihrer Kunden vor unzähligen Bedrohungen und Sicherheitslücken zu schützen. Dabei forcieren strikte Vorschriften hohe Investitionen in die eigenen Prozesse und Systeme. Mit Investitionen alleine ist es allerdings noch nicht getan. IT-Sicherheit selbst ist ein Prozess, der laufend überwacht und verbessert werden muss, um den Risiken moderner Angriffe entgehen zu können. Unternehmen müssen sich dem bewusst sein und auch die notwendigen Schritte dazu vornehmen.

Einen zentralen Punkt zur Durchsetzung von Abwehrmechanismen gibt es dabei nicht. Aufgrund der Vielseitigkeit unterschiedlicher Software-Installationen kann nicht jede externe Bedrohung abgewehrt werden, bevor sie ein System erreicht. Viele moderne Angriffe versuchen mittlerweile auf Anwendungsebene Zugriff auf Applikationen und Server zu erlangen oder deren Verfügbarkeit zu beeinträchtigen. Moderne Applikationen verfügen zwar über aktuelle Sicherheitstechnologien und stützen sich darüber hinaus auf Frameworks, die die Sicherheit teilweise zusätzlich erhöhen. Die konstante weiterentwicklung moderner Technologien und Systeme erfordert allerdings auch, dass die Applikationen, die diese Technologien verwenden und mit den Systemen kommunizieren, laufend gewartet werden.

Innerhalb einer Applikation auf diese große Anzahl von Bedrohungen reagieren zu können ist dabei schwer vorstellbar und dürfte in den meisten Fällen nicht im Verhältnis zum effektiven Nutzen der Applikation stehen. Trotzdem ist es erstrebenswert, bereits applikationsseitig auf die offensichtlichsten Sicherheitslücken Rücksicht zu nehmen, wie beispielsweise die Verschlüsselung von Passwörtern.

Wie beschrieben wurde, kann eine Web Application Firewall dabei helfen, die Angriffsfläche zu reduzieren, einerseits über vorgefertigte Konfigurationen und andererseits über eigens erstellte Regeln oder Erweiterungen. Im praktischen Beispiel in Kapitel 4 wurde dies anhand einer Implementierung für die Validierung von Cookies gezeigt. Die Implementierung wäre applikationsseitig vermutlich schneller gegangen. Allerdings stellt die Verwendung von ModSecurity dar, dass eine solche Anforderung auch ohne Änderung an der Applikation durchgeführt werden kann. Das Prinzip und die Funktionsweise von

WAF konnten dabei mit ModSecurity gut dargestellt werden und ein direkter Vergleich mit einer kommerziellen Lösung hätte in diesem Kontext keinen Mehrwert gebracht.

Web Application Firewalls bringen bereits eine breite Palette an Funktionen mit sich und es sollte davon absehen werden, das Rad neu zu erfinden. Eher sollte man auf eine Kombination mehrerer Sicherheitsmechanismen setzen. Erhöht sich die Anzahl an Systemen oder deren Komplexität, kann allerdings auch der Einsatz einer WAF schnell an Komplexität gewinnen. Bei einer hohen Anzahl an definierten Regeln, ist es schwer den Überblick zu behalten. Darüber hinaus steht man vor der Herausforderung, die definierten Regeln korrekt für alle betroffenen Systeme anzupassen und ein geeignetes Mittelmaß zu finden.

Mit dem bereitgestellten Konzept wurde allerdings gezeigt, dass der Einsatz der Open-Source WAF ModSecurity auch einfach applikationsseitig bereitgestellt und somit dezidiert für die Absicherung einer spezifischen Applikation verwendet werden kann. Bei großen verteilten Systemen kann dies jedoch auch schnell aus dem Ruder laufen und den Wartungsaufwand erheblich erhöhen. Darüber hinaus erfordern solche Systeme gegebenenfalls auch die Verwendung von IDS oder IPS Lösungen, um gegen die hohe Anzahl Bedrohungen gerüstet zu sein.

Für große Systeme scheinen kommerzielle Lösung von Web Application Firewalls als Managed Service eine geeignete Alternative gegenüber dem Betrieb von selbst gewarteten WAF zu sein. Kleinere Systeme hingegen können bereits von dem Betrieb einer selbst gewarteten Open-Source Lösung profitieren. Sowohl kommerzielle als auch Open-Source Lösungen bringen dabei eine Vielzahl an Funktionen mit sich, mit denen auf die gängigsten Bedrohungen und auch auf die eigenen applikationsseitigen Anforderungen effektiv eingegangen werden kann. Von dem Versuch, die Konzepte, die Web Application Firewalls bereitstellen, auf Seiten der Applikation selbst zu implementieren, sollte abgesehen werden. Der Aufwand, um das Ausmaß an Bedrohungen abwehren zu können, steht dabei nicht im Verhältnis zu den notwendigen Aufwendungen, eine Web Application Firewall initial und applikationsspezifisch bereitzustellen.

ABBILDUNGSVERZEICHNIS

2.1 Monolith vs. Microservice Design 12
2.2 Struktur einer SOAP Nachricht 14

3.1 Der Ritter der Firewall 34
3.2 Netzwerk-Firewall 35
3.3 OSI-Schichten 38
3.4 OWASP Top 10 39
3.5 Objektmodell eines HTTP Requests 45
3.6 Teile eineer SQL-Injection 47
3.7 Dreibeinige WAF 50
3.8 WAF Sandwich 51
3.9 Transparent Proxy WAF 52
3.10 Forard Proxy WAF 53
3.11 WAF Monitoring Mode 54
3.12 WAF Server Mode 54
3.13 Der Ritter der Firewall: Durchbruch 55
3.14 Gartner WAF Magic Quadrant 60

4.1 HTTP-Cookie Ablauf 72
4.2 Konzept Cookie Signatur 73

TABELLENVERZEICHNIS

3.1 Quantifizierung SQL-Injection 47

LISTINGS

2.1 Beispiel XML Objekt . . . 15
2.2 Beispiel JSON Objekt . . . 15
2.3 Einfaches SQL Select Statement . . . 18
2.4 Beispiel SQL Injection . . . 19
2.5 GraphQL-Anfrage [1] . . . 25
2.6 GraphQL Antwort [2] . . . 26

4.1 Cookie Aufbau [3] . . . 72
4.2 Cookie Controller . . . 75
4.3 Curl GET Anfrage . . . 76
4.4 ModSecurity Rules Engine Aktivierung . . . 77
4.5 ModSecurity Beispiel Regel . . . 77
4.6 ModSecurity Regel Match . . . 78
4.7 ModSecurity Audit Log . . . 79
4.8 ModSecurity Cookie erkennen . . . 80
4.9 ModSecurity Cookie Blockieren . . . 81
4.10 ModSecurity Cookie Blockieren Log . . . 81
4.11 ModSecurity Cookie Wert Extrahieren . . . 82
4.12 ModSecurity Cookie Wert Extrahieren und blockieren . . . 82
4.13 ModSecurity Hash berechnen . . . 83
4.14 ModSecurity Hash als Cookie übergeben . . . 83
4.15 ModSecurity Cookie Hash Ergebnis . . . 84
4.16 ModSecurity Cookie Übermittlung prüfen . . . 84
4.17 ModSecurity Cookie Übermittlung blockieren . . . 85
4.18 ModSecurity Cookie Übeermittlung zulassen . . . 86
4.19 ModSecurity Regel: Signatur Vergleich . . . 87
4.20 ModSecurity Signatur Übereinstimmung . . . 88
4.21 ModSecurity Signatur Fehler . . . 89
4.22 ModSecurity Signatur Fehler . . . 89

LITERATURVERZEICHNIS

Anders, D. (o. J.). *Introduction to Web Application Firewalls.* Zugriff am 05.04.2022 auf https://owasp.org/www-pdf-archive/Intro-WebApplicationFirewalls.pdf

Anderson, B. (2020). *3 Dangerous Cross-Site Scripting Attacks of the Last Decade.* Zugriff am 09.01.2022 auf https://readwrite.com/2020/11/30/3-dangerous-cross-site-scripting-attacks-of-the-last-decade/

Babu, M. (2019, April). *Top 5 Challenges Securing Applications with Web Application Firewalls.* Zugriff am 24.04.2022 auf https://www.contrastsecurity.com/security-influencers/top-5-challenges-securing-applications-with-web-application-firewalls

Bedner, M. & Ackermann, T. (2010). Schutzziele der IT-Sicherheit. *Datenschutz und Datensicherheit - DuD*, *34* (5), 323–328.

Bentz, V. (2019). *Mit diesen Datenmengen werden wir in Zukunft rechnen.* Zugriff am 10.12.2021 auf https://www.brandmauer.de/blog/it-strategie/mit-diesen-datenmengen-werden-wir-in-zukunft-rechnen

Bhardwaj, R. (o. J.). *FORWARD PROXY VS REVERSE PROXY.* Zugriff am 07.04.2022 auf https://ipwithease.com/forward-proxy-vs-reverse-proxy/

Blatz, J. (2007). *CSRF: Attack and Defense* (Bericht). McAfee® Foundstone® Professional Services.

Braue, D. (2021, Juni). *10 Hot MDR Companies To Watch In 2021.* Zugriff am 01.04.2022 auf https://cybersecurityventures.com/10-hot-mdr-companies-to-watch-in-2021/

Cezar, A., Cavusoglu, H. & Raghunathan, S. (2010). Competition, Speculative Risks, and IT Security Outsourcing. In T. Moore, D. Pym & C. Ioannidis (Hrsg.), *Economics of information security and privacy* (S. 301–320). Boston, MA: Springer US.

Cezar, M. (2018, Januar). *How to Change Apache HTTP Port in Linux.* Zugriff am 04.04.2022 auf https://www.tecmint.com/change-apache-port-in-linux/

Cyber Defense Media Group. (2021, Februar). *Top 100 Managed Security Service Providers (MSSPs).* Zugriff am 01.04.2022 auf https://www.cyberdefensemagazine.com/top-100-managed-security-service-providers-mssps/

Dermann, M., Dziadzka, M., Hemkemeier, B., Hoffmann, A., Meisel, A., Rohr, M. & Schreiber, T. (2008). *Best Practices: Use of Web Application Firewalls* (Bericht). OWASP Foundation.

D'Hoinne, J., Hils, A., Kaur, R. & Watts, J. (2020, Oktober). *Magic Quadrant for Web Application Firewalls* (techreport). Gartner Inc.

Endraca, A., King, B., Nodalo, G., Maria, M. S. & Sabas, I. (2013, Dezember). Web Application Firewall (WAF). *International Journal of e-Education, e-Business, e-Management and e-Learning*, *3* (6), 451-455.

Ferguson, D. (2017). *Cross-Site Request Forgery: What Happened to the Sleeping Giant?* Zugriff am 12.01.2022 auf https://blog.qualys.com/product-tech/2017/01/26/cross-site-request-forgery-what-happened-to-the-sleeping-giant

Folini, C. & Ristic, I. (2017). *ModSecurity Handbook* (2. Aufl.). London: Feisty Duck.

Freda, A. (2021, Februar). *Was ist ein Zero-Day-Angriff?* Zugriff am 03.04.2022 auf https://www.avast.com/de-de/c-zero-day

Galluccio, E., Casell, E. & Lombari, G. (2020). *SQL Injection Strategies*. Birmingham: Packt Publishing.

Gillis, A. (2018, August). *IT incident management.* Zugriff am 22.03.2022 auf https://www.techtarget.com/searchitoperations/definition/IT-incident-management

Gonzalez, R., Gasco, J. & Llopis, J. (2010, 03). Information systems outsourcing reasons and risks: A new assessment. *Industrial Management and Data Systems*, *110*, 284-303. doi: 10.1108/02635571011020359

Grunwald, L. & Herz, A. (2017, Juli). *Intrusion-Detection- und -Prevention-Systeme.* Zugriff am 22.03.2022 auf https://www.heise.de/select/ix/2017/7/1499605428538281

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J., Nielson, H., Karmarkar, A. & Lafon, Y. (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* Zugriff am 05.01.2022 auf https://www.w3.org/TR/2007/REC-soap12-part1-20070427/

Güttich, G. & Schmitz, P. (2017, Oktober). *Grundlagen der Next Generation Firewall (NGFW).* Zugriff am 03.04.2022 auf https://www.security-insider.de/grundlagen-der-next-generation-firewall-ngfw-a-648098/

Haltdos Inc. (2020, März). *What is the Difference between Whitelisting WAF vs Blacklisting WAF?* Zugriff am 05.04.2022 auf https://www.haltdos.com/blog/what-is-the-difference-between-whitelisting-waf-vs-blacklisting-waf

Hirschheim, R., Heinzl, A. & Dibbern, J. (2020). *Information Systems Outsourcing: The Era of Digital Transformation* (5. Aufl.). Springer, Cham.

Hoffman, A. (2020). *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. Sebastopol: O'Reilly Media.

IBM Coorporation. (2021, 12.). *The Structure of a SOAP Message.* Zugriff am 05.01.2022 auf https://www.ibm.com/docs/en/integration-bus/10.0?topic=

soap-structure-message

Imperva Inc. (o. J.). *Reverse Proxy vs Forward Proxy.* Zugriff am 07.04.2022 auf https://www.imperva.com/learn/performance/reverse-proxy/

Ingham, K., Consulting, K. & Forrest, S. (2002). A history and survey of network firewalls. *ACM Journal Name*, 1-42.

IONOS SE. (2019). *XSS/Cross-Site-Scripting unterbinden und Sicherheitslücken schließen.* Zugriff am 08.01.2022 auf https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-xss-bzw-cross-site-scripting/

itanius informatik GmbH. (o. J.). *User Manual WebCastellum Web Application Firewall* (Bericht).

Kelley, D. (2021, April). *MDR vs. MSSP: Why it's vital to know the difference.* Zugriff am 01.04.2022 auf https://www.techtarget.com/searchsecurity/tip/MDR-vs-MSSP-Why-its-vital-to-know-the-difference

Kiarie, J. (2020, Juni). *How to Install Apache with Virtual Hosts on Debian 10.* Zugriff am 04.06.2022 auf https://www.tecmint.com/install-apache-with-virtual-hosts-on-debian-10/

Kohnfelder, L. (2021). *Designing Secure Software*. San Francisco: No Starch Press.

Kolokotronis, N. & Shiaeles, S. (2021). *Cyber-Security Threats, Actors, and Dynamic Mitigation*. Boca Raton: CRC Press.

L7 Defense Ltd. (2021, September). *Why WAF Solutionsare Limited and Fails to Keep APIs Completely Safe?* Zugriff am 24.04.2022 auf https://www.l7defense.com/why-waf-solutions-cannot-keep-apis-completely-safe/

Lakhno, V., Blozva, A. A., Kasatkin, D., Chubaievskyl, V., Shestak, Y., Tyshchenko, D. & Brzhanov, R. (2022). EXPERIMENTAL STUDIES OF THE FEATURES OF USING WAF TO PROTECT INTERNAL SERVICES IN THE ZERO TRUST STRUCTURE. In *Journal of theoretical and applied information technology* (Bd. 100, S. 705-721).

Leenes, R., van Brakel, R., Gutwirth, S. & De Hert, P. (2017). *Data Protection and Privacy: (In)visibilities and Infrastructures*. Cham: Springer International Publishing AG.

Lord, N. (2018, September). *What is Advanced Threat Detection?* Zugriff am 03.03.2022 auf https://digitalguardian.com/blog/what-advanced-threat-detection

Luber, S. & Donner, A. (2018, August). *Was ist NAT (Nerwork Address Translation)?* Zugriff am 07.04.2022 auf https://www.ip-insider.de/was-ist-nat-network-address-translation-a-663954/

Luber, S. & Schmitz, P. (2018a, Januar). *Was ist eine DMZ (Demilitarized Zone)?* Zugriff am 03.03.2022 auf https://www.security-insider.de/was-ist-eine-dmz-demilitarized-zone-a-677267/

Luber, S. & Schmitz, P. (2018b, Juni). *Was ist ein Hardware-Sicherheitsmodul*

(HSM)? Zugriff am 01.05.2022 auf https://www.security-insider.de/was-ist-ein-hardware-sicherheitsmodul-hsm-a-727090/

MacVittie, L. (2008, Juli). *4 reasons not to use mod-security.* Zugriff am 01.05.2022 auf https://community.f5.com/t5/technical-articles/4-reasons-not-to-use-mod-security/ta-p/284629

Malwarebytes Inc. (o. J.). *SQL injection.* Zugriff am 10.01.2022 auf https://www.malwarebytes.com/sql-injection

Moles, J. (2021, Juli). *TCP Resets (RST): Attack or Defender Containment Method?* Zugriff am 07.04.2022 auf https://www.extrahop.com/company/blog/2021/tcp-resets-attack-or-defender-containment/

Mozilla Corporation. (2022, April). *Using HTTP cookies.* Zugriff am 27.05.2022 auf https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

Nayak, P., Ray, N. & Ravichandran, P. (2022). *IOT APPLICATIONS, SECURITY THREATS, AND COUNTERMEASURES*. Boca Raton: CRC Press.

Novikov, I. (2019, Oktober). *Legacy WAF Issues.* Zugriff am 24.04.2022 auf https://lab.wallarm.com/traditional-wafs-tell-a-lot-about-consumer-adoption/

Omelchenko, I. (2021). *Monolithic vs. Microservices - the choice that defines the whole development process.* Zugriff am 10.12.2021 auf https://clockwise.software/blog/monolithic-architecture-vs-microservices-comparison/

OWASP Foundation. (2020, Juli). *XML External Entity (XXE) Processing.* Zugriff am 24.04.2022 auf https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

OWASP Foundation. (2021a, Oktober). *OWASP Top Ten.* Zugriff am 03.04.2022 auf https://owasp.org/www-project-top-ten/

OWASP Foundation. (2021b, Dezember). *Session hijacking attack.* Zugriff am 27.05.2022 auf https://owasp.org/www-community/attacks/Session_hijacking_attack

OWASP Foundation. (2022, Februar). *Cross Site Scripting (XSS).* Zugriff am 08.01.2022 auf https://owasp.org/www-community/attacks/xss/

Porcello, E. & Banks, A. (2018). *Learning GraphQL*. Sebastopol: O'Reilly Media.

PortSwigger Ltd. (o. J.). *Reflected XSS.* Zugriff am 08.01.2022 auf https://portswigger.net/web-security/cross-site-scripting/reflected

Qualys Inc. (2011). *IronBee Open Source Web Application Firewall: Building a universal web application firewall engine* (techreport). Qualys Inc.

Russel, C. (2018). *Web Application Firewalls: Securing Modern Web Applications* (C. Allen, Hrsg.). Sebastopol, CA: O'Reilly Media.

Saeed, Y. (2021, Mai). *How To Configure Apache as a Reverse Proxy For Ubuntu/Debian.* Zugriff am 04.06.2022 auf https://blog.containerize.com/2021/05/21/how-to

-configure-apache-as-a-reverse-proxy-for-ubuntudebian/

Sampaio, D. & Bernardino, J. (2017). Evaluation of Firewall Open Source Software. In *Proceedings of the 13th international conference on web information systems and technologies.*

Schaar, P. (2020). Datenschutz und Internet – Es ist kompliziert! *Informatik Spektrum*, *43* (3), 179-185.

Schubert, C. J. (2021, Februar). *Port Mirroring: Ist ein TAP wirklich besser als ein SPAN Port?* Zugriff am 07.04.2022 auf https://www.magellan-net.de/de/aktuelles/blog/details/port-mirroring-ist-ein-tap-wirklich-besser-als-ein-span-port/

Sermpinis, T. (2017). *Hakin9 Workshops: Bypassing Web Application Firewalls* (Bd. 12; J. Kretowicz, M. Sienicka, M. Strzelec & A. Kondzierska, Hrsg.) (Nr. 6). Warszawa: Hakin9 Media Sp. z o.o.

Smith, M. (2021, April). *Web Application Firewall Rule Internals.* Zugriff am 05.04.2022 auf https://www.youtube.com/watch?v=Qknpik-KqRA

Spherical Defence AI Inc. (2018, Oktober). *Why WAF's Won't Work?* Zugriff am 24.04.2022 auf https://sphericaldefence.com/why-wafs-dont-work/

Stewart, J. M. & Kinsey, D. (2020). *Network Security, Firewalls, and VPNs,* (Third Aufl.). Burlington, MA: Jones & Bartlett Learning.

Subramanian, H. & Raj, P. (2019). *Hands-On RESTful API Design Patterns and Best Practices*. Birmingham: Packt Publishing.

Sundar, V. (2021, Dezember). *How Web Application Firewall Works?* Zugriff am 05.04.2022 auf https://www.indusface.com/blog/how-web-application-firewall-works/

The GraphQL Foundation. (o. J.). *Star Wars API.* Zugriff am 22.01.2022 auf https://graphql.org/swapi-graphql

Velasco, R. (2018, November). *Agile protection: above and beyond the WAF.* Zugriff am 24.04.2022 auf https://hdivsecurity.com/bornsecure/agile-protection-above-and-beyond-the-waf/

Vodopyan, E. (2022, Januar). *Understanding Insider Threats: Definition and Examples.* Zugriff am 03.04.2022 auf https://blog.netwrix.com/2021/06/23/insider-threat/

Voigt, P. & von dem Bussche, A. (2017). *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Cham: Springer, Cham.

Wendzel, S. (2018). *Grundlagen der IT-Sicherheit*. Wiesbaden: Springer Fachmedien Wiesbaden.

Wickett, J. (2018, März). *Three Ways Legacy WAFs Fail.* Zugriff am 24.04.2022 auf https://www.signalsciences.com/blog/three-ways-wafs-fail/