WHU−Otto Beisheim School of Management

Assignment III

Security Aspects in a Web Framework

Software Development for Start-Up Entrepreneurs

SOFTLANG Team, Universität Koblenz-Landau

Submission Date: April 17th, 2014

Submitted to: Prof. Dr. Ralf Lämmel

By

Daniel Deutsch

Schulstraße 10, 56179 Vallendar

Born May 11th, 1991 in Bad Ems

Justus Hammer

Heerstraße 60, 56179 Vallendar

Born May 1st, 1992 in Duisburg

Christina Heise

Burgplatz 1b, 56179 Vallendar

Born December 25th, 1989 in Hamburg

Inhaltsverzeichnis

1. Introduction

Im Folgenden werden wir das Thema Sicherheitsrisiken für Webanwendungen behandeln. Dafür werfen wir einen Blick auf die in der Vorlesung geschriebene *Polls* Anwendung, speziell mit Bezug auf die verwendete *REST API*, welche *Django*, dem in *Python* geschriebenen qülloffenen Web-Framework, unterliegt. Im weiteren Verlauf zeigen wir die zehn häufigsten Sicherheitsrisiken auf, erläutern wie der Schutz vor solchen Bedrohungen zu gestalten ist und stellen sie in Bezug mit Django's integrierten Sicherheitsmaßnahmen.

2. OWASP

Das Open Web Application Security Project (OWASP) ist eine offene Community die durch kostenloses bereitstellen von Artikeln, Vorgehensweisen, Dokumentationen und Werkzeugen Unternehmen und Organisationen bei der Entwicklung, dem Kauf und der Wartung sicherer Anwendung unterstützt. Die OWASP Top 10 repräsentiert die 10 größten Risiken für die Anwendungssicherheit. Sicherheisexperten auf der ganzen Welt teilen ihr Fachwissen zur Erstellung der Liste und der daraus resultierenden Gefahrensensibilisierung von Organisationen. An OWASP als Non-Profit-Organisation sind Firmen, Bildungseinrichtungen und Einzelpersonen aus aller Welt beteiligt und steht nicht mit Technologiefirmen in Verbindung, obgleich es den bedachten Einsatz von Sicherheitstechnologie unterstützt. Die Verbindungen werden vermieden, um frei von organisationsseitigen Zwängen zu sein. Dadurch wird es leichter, unvoreingenommene, praxisnahe und wirtschaftliche Informationen über Applikationssicherheit bereitzustellen.
Die Liste der Top 10 Sicherheitsrisiken für Webanwendungen wird regelmäßig aktualisiert, da dieses Feld sehr dynamisch ist und stetig neü Risiken entdeckt beziehungsweise anhand ihrer Relevanz neu bewertet werden. Momentan sieht die Liste wie folgt aus:

Top 10 Risiken für Anwendungssicherheit

A1 - Injection

A2 - Cross-Site Scripting (XSS)

A3 - Fehler in Authentifizierung und Session Management

A4 - Unsichere direkte Objektreferenzen

A5 - Cross-Site Reqüst Forgery (CSRF)

A6 - Sicherheitsrelevante Fehlerkonfiguration

A7 - Kryptographisch unsicher Speicherung

A8 - Mangelhafter URL-Zugriffsschutz

A9 - Unsichere Absicherung der Transportschicht

A10 - Ungeprüfte Um- und Weiterleitung

3. Sicherheitsrisiko

Schwachstellen in der Programmierung von Webanwendungen werfen Sicherheisrisiken für die Anwendungen auf. Angreifer können diese nutzen um Angriffswege innerhalb der Anwendung zu verwenden, meist geschieht dies um sich wirtschaftliche Vorteile zu generieren oder sonstigen Schaden, bei Unternehmen, Organisationen und Privatpersonen anzurichten. Das folgende Diagramm erläutert den Weg einer Bedrohungsqülle schematisch.

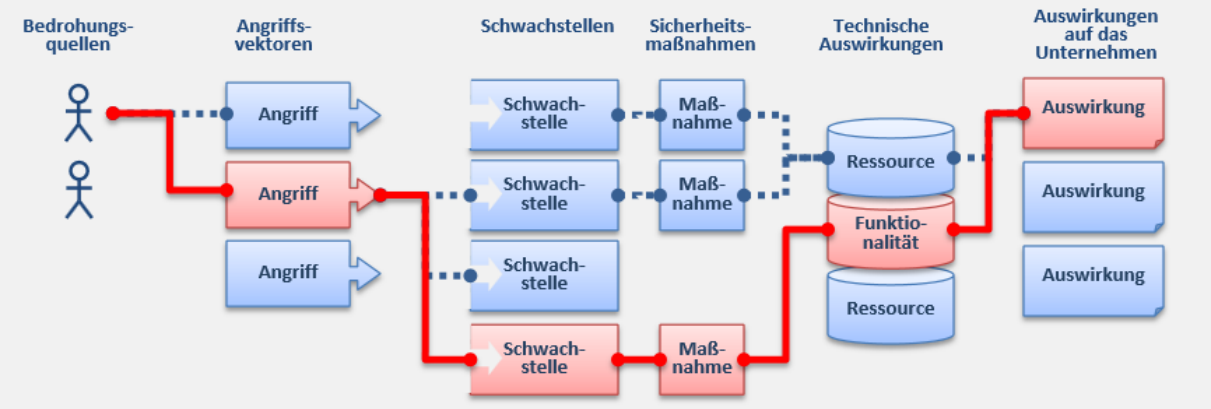


Abbildung 1

Während manche Wege einfach zu entdecken und zu benutzen sind, sind andere sehr schwierig zu finden. Durch einer Kombination der jeweiligen technischen Auswirkungen und den Auswirkungen auf den Geschäftsbetrieb mit den Wahrscheinlichkeit der Bedrohungsqülle, der Angriffsvektoren und den Sicherheitsschwachstellen, kann man das individülle Gesamtrisiko abschätzen.

4. OWASP Testing Guide

Ein Testing Guide für Sicherheitsrisiken von OWASP wird, wie auch die Top 10 Liste der Bedrohungen selbst, regelmäßig aktualisiert und in Form eines über dreihunderseitigen Dokuments veröffentlicht. Folgende Übersicht gibt einen strukturierten Überblick über die verschiedenen Möglichkeiten des Testens.

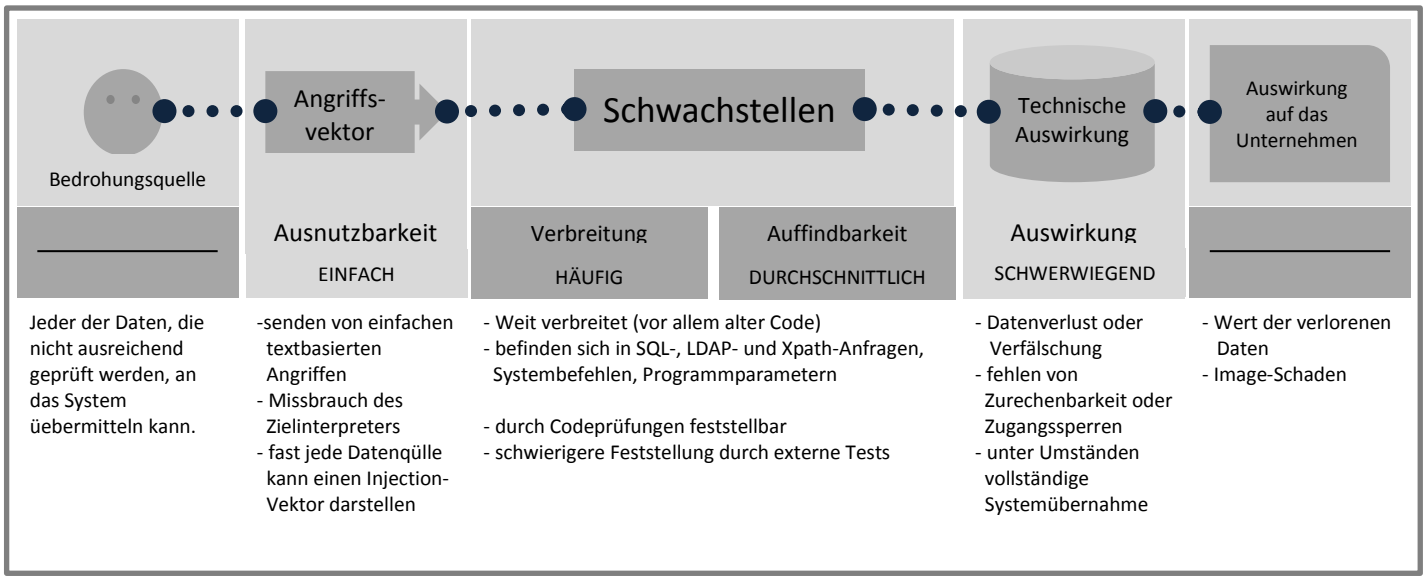
Information Gathering	Configuration Management Testing	Authentication Testing	Session Management	Authorization Testing	Business logic testing	Data Validation Testing	Denial of Service Testing	Web Services Testing	AJAX Testing
Spiders, Robots and Crawlers	SSL/TLS Testing	Credentials transport over an encrypted channel	Testing for Session Management Schema	Testing for Path Traversal	Testing for business logic	Testing for Reflected Cross Site Scripting	Testing for SQL Wildcard Attacks	WS Information Gathering	AJAX Vulnerabilities
Search Engine Discovery	DB Listener Testing	Testing for user enumeration	Testing for Cookies attributes	Bypassing authorization schema		Testing for Stored Cross Site Scripting	Locking Customer Accounts	Testing WSDL	AJAX Testing
Identify application entry points	Infrastructure Configuration Management Testing	Testing for Guessable (Dictionary) User Account	Testing for Session Fixation	Testing for Privilege Escalation		Testing for DOM based Cross Site Scripting	Testing for DoS Buffer Overflows	XML Structural Testing	
Testing for Web Application Fingerprint	Application Configuration Management Testing	Brute Force Testing	Testing for Exposed Session Variables			Testing for Cross Site Flashing	User Specified Object Allocation	XML content-level Testing	
Application Discovery	Testing for File Extensions Handling	Testing for bypassing authentication schema	Testing for CSRF			SQL, LDAP, ORM, XML, SSI, XPATH, SMTP, Code Injection	User Input as a Loop Counter	HTTP GET parameters/ REST Testing	
Analysis of Error Codes	Old, backup and unreferenced files	Testing for vulnerable password and reset				OS Commanding	Writing Data to Disk	Naughty SOAP attachments	
	Admin Interfaces	Logout and Browser Cache Management				Buffer overflow	Failure to Release Resources	Replay Testing	
	Testing for HTTP Methods and XST	Testing for CAPTCHA				Incubated vulnerability Testing	Too Much Data in Session		
		Testing Multiple Factors Authentication				Testing for HTTP Splitting/ Smuggling			
		Testing for Race Conditions							

Abbildung 2

Im folgenden Teil werden die Top 10 Sicherheitsrisiken für Webanwendungen im Detail besprochen.

5.1. Injection

Injection-Schwachstellen entstehen, wenn nicht vertraünswürdige Daten als Teil eines Kommandos oder einer Abfrage von einem Interpreter verarbeitet werden, sodass ein Angreifer diese dann so manipulieren kann, dass er Kommandos ausführen kann und unautorisiert auf Daten zugreifen kann.



Verwundbarkeit

Um herauszufinden ob man durch Injection verwundbar ist, ist die Unterscheidung der Interpreter zwischen Eingabedaten und Befehle zu prüfen. Code-Prüfung mithilfe von Code-Analyse-Werkzeugen helfen bei der Identifizierung des Interpretergebrauchs und des Datenflusses. Penetrationstests können Schwachstellen durch deren Ausnutzung bestätigen.

Prävention

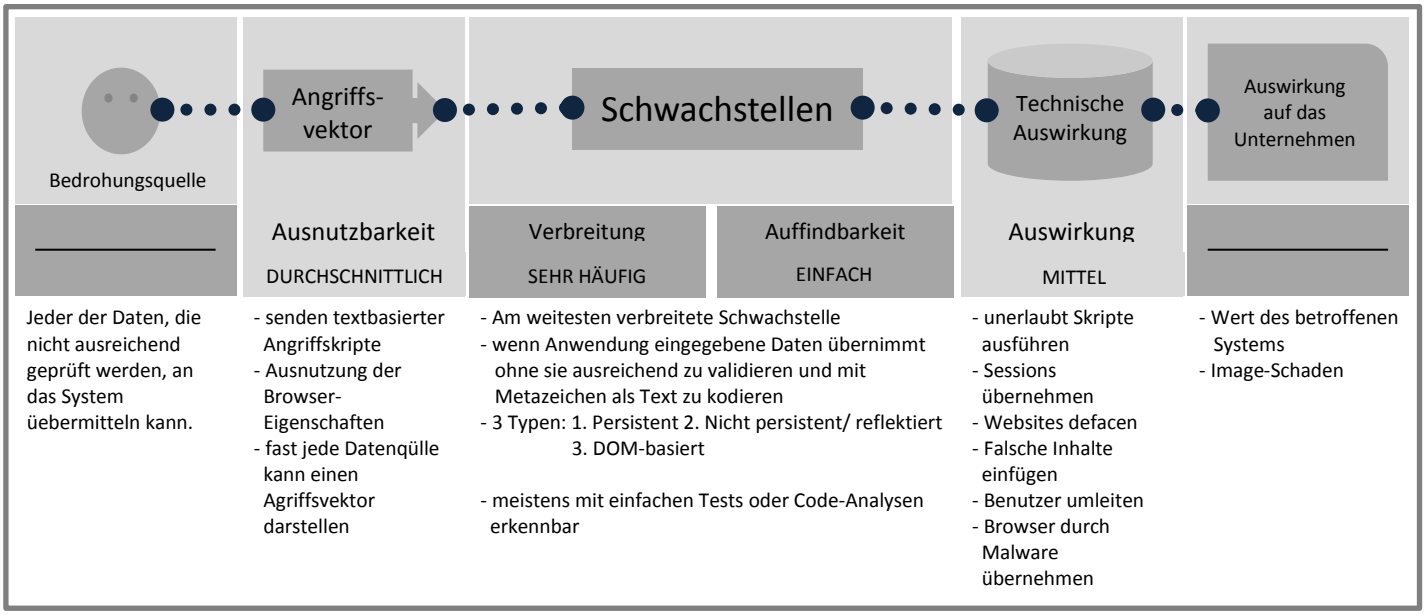
- konsequente Trennung von Eingabedaten und Befehlen
- Nutzung einer sicheren API, ansonsten Metazeichen unter Berücksichtigung der jeweiligen Syntax entschärfen

Sicherheitsmaßnahme in Django

- Djangos ORM Schicht schützt durch automatisches Entschärfen
- Ausnahme I: Verwendung der Funktion extra()
- Ausnahme II: Verwendung der Low-Level APIs

5.2. Cross-Site Scripting (XSS)

Diese treten auf, wenn nicht vertraünswürdige Daten ohne Validierung und Kodierung entgegengenommen werden und an einen Webbrowser gesendet werden. Dadurch können Angreifer Benutzersitzungen übernehmen, Seiteninhalte manipulieren oder Weiterleitungen auf bösartige Seiten vornehmen.



Verwundbarkeit

Eine Validierung aller vom Benutzer eigegebenen Daten, die zurück an den Browser gesendet werden, und die spezielle Kodierung der Ausgabedaten sodass eingegebener Text nicht als aktiver Inhalt ausführbar ist, sorgt für Sicherheit bei XSS. Automatisierte Test-Tools in Verbindung mit manüllem Code Review sowie manüllen Penetrationstests macht es möglich XSS aufzudecken.

Prävention

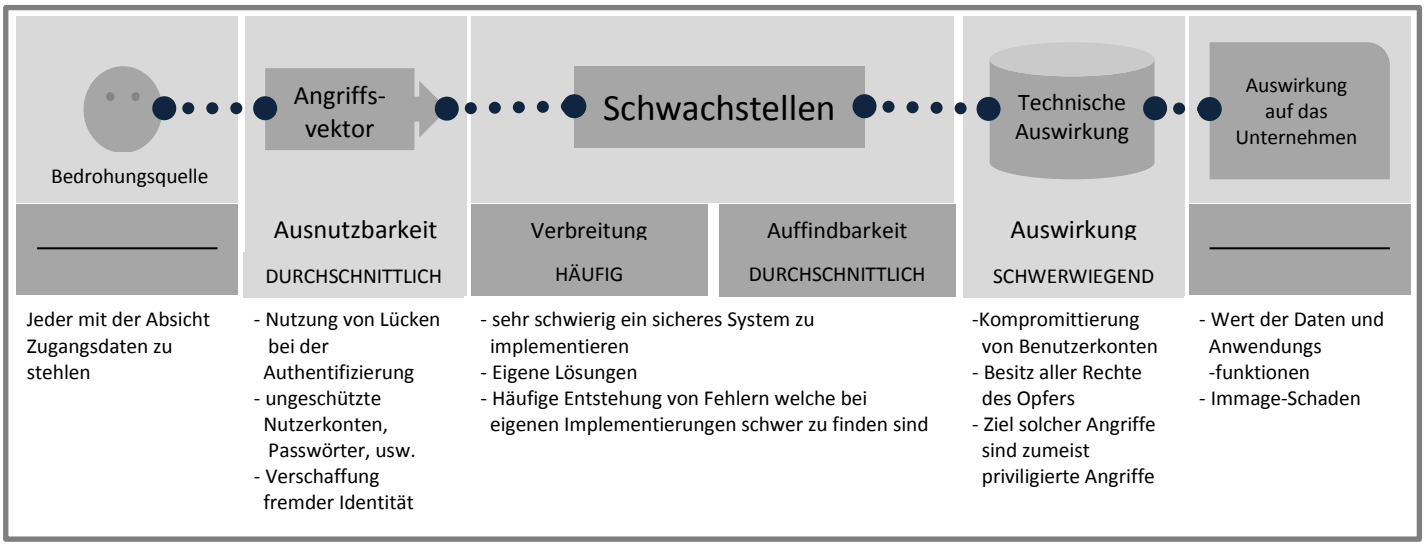
- nicht vertrauenswürdige Daten von aktiven Browserinhalten trennen
- nicht vertrauenswürdige Metazeichen entsprechend escapen

Sicherheitsmaßnahme in Django

- Djangos Template-System verhindern XSS durch standardmäßige automatische Parameterwertentschärfung
- Werte, die nicht entschärft werden sollen, können explizit gekennzeichnet werden

5.3. Fehler in Authentifizierung und Session-Management

Authentifizierungs- und Session-Managementfunktionen werden häufig nicht richtig implementiert. Diese Fehler erlauben es Angreifern Passwörter und Sessiontokens zu manipulieren und so auszunutzen, dass sie die Identität anderer Benutzer annehmen können.



Verwundbarkeit

Ressourcen wie Benutzerkennungen und Passwörter sind durch geschützte Speicherung (z.B. Hashverfahren) und das Unmöglichmachen von Erraten und Überschreiben der Benutzernamen zu schützen. Session IDs sollten unter Inbetrachnahme der Sichtbarkeit in der URL sowie das Auslaufen bzw. Verändern bei Anmeldung/Abmeldung benutzt werden.

Prävention

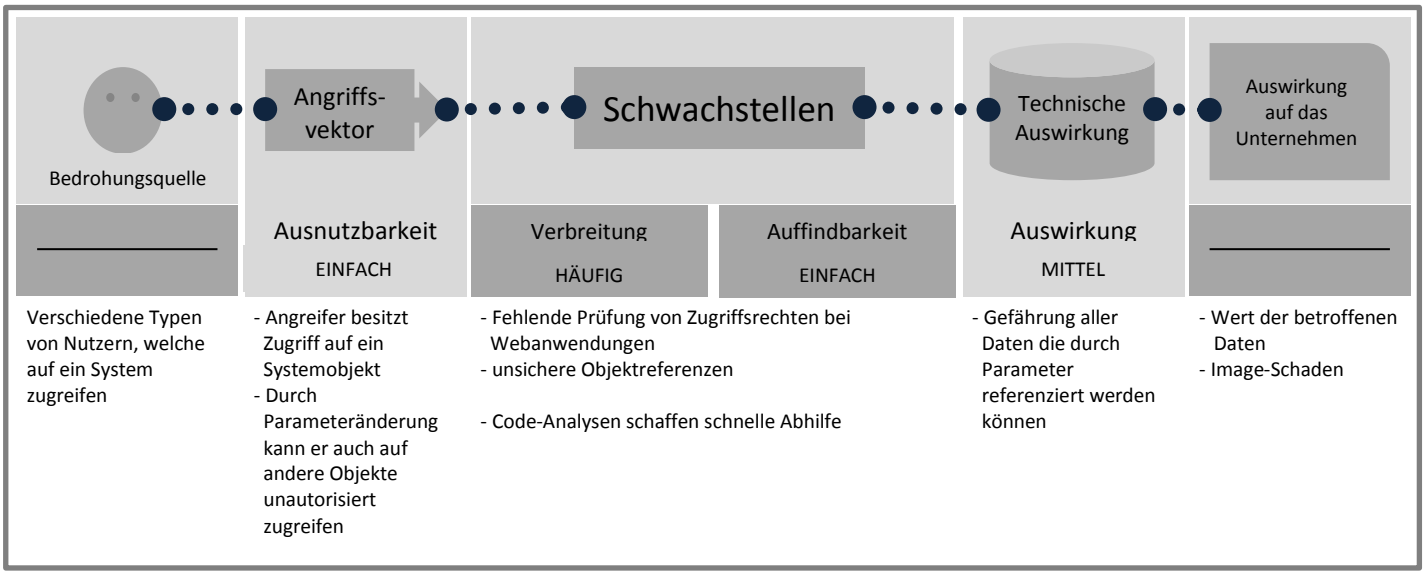
- Unternehmensseitiges Bereitstellen zentraler Mechanismen für die wirksame Authentifizierung und Session-Management
- Ressourcen für die Implementierung der Vermeidung von XSS, und Session-ID Diebstählen

Sicherheitsmaßnahme in Django

- Django bietet implementierte Session Frameworks an
    - Cookie enthält lediglich die Session ID
    - Daten der Session befinden sich in der Backend-DB
- Session Ids werden außerdem gehashed
- Standardmäßig jedoch unsicher

5.4. Unsichere direkte Objektreferenzen

Diese treten auf, wenn Referenzen zu internen Implementierungsobjekten, wie Dateien, Ordner oder Datenbankschlüssel von außen zugänglich sind. Dadurch können Angreifer unautorisiert Zugriff auf Daten erlangen.



Verwundbarkeit

Alle Referenzen sollten über passende Abwehrmechanismen verfügen; direkte Referenzen auf eingeschränkte Ressourcen müssen durch die Anwendung auf Nutzerauthorisierung, indirekte Objektreferenzen müssen bei Zuordnung auf entsprechende direkte Referenzen auf Nutzerautorisierte Werte beschränkt sein. Code Reviews und Testing sind für direkte Referenzen geeignet.

Prävention

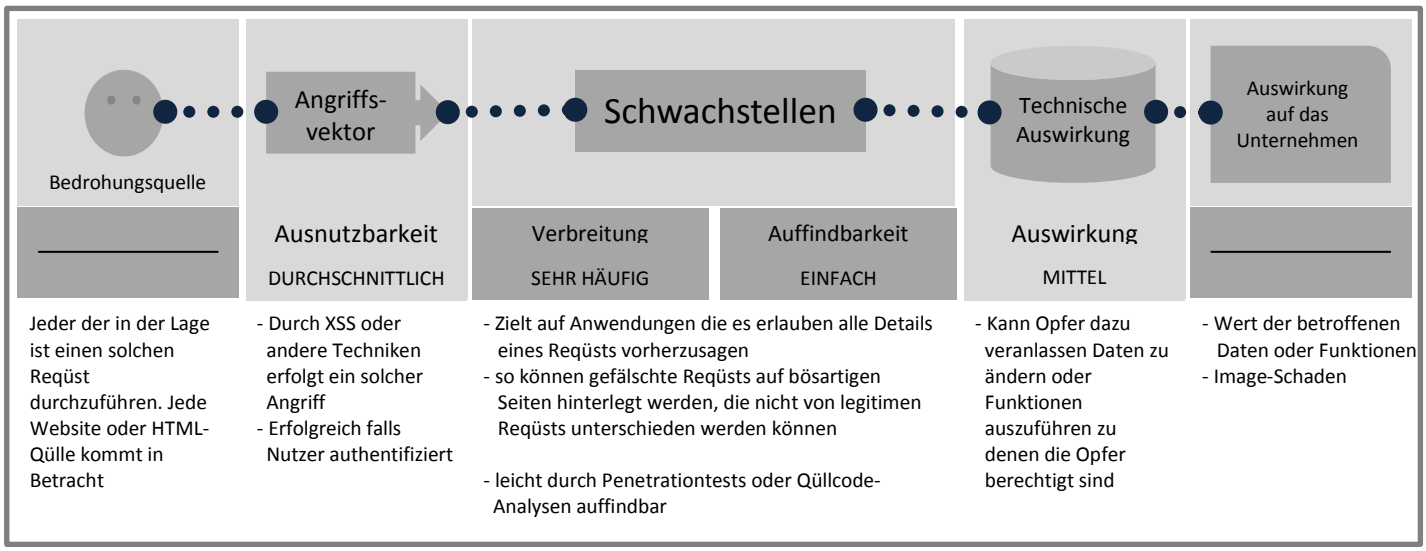
- Schutz eines jeden Objektes durch indirekte Objektreferenzen sowie das Prüfen der Zugriffe

Sicherheitsmaßnahme in Django

- Wenn man die Objekt ID nicht verwenden möchte bietet Django ein sogenanntes “slug”-Feld an
- Views können dazu verwendet werden, Nutzern bestimmte Rechte zu definieren
- Zugriffserlaubnis für Objekte muss jedoch manüll überprüft werden

5.5. Cross-Site Reqüst Forgery (CSRF)

Bei einem solchen Angriff werden manipulierte HTTP-Reqüsts durch den Browser eines angemeldeten Benutzers an die verwundbare Anwendung gesendet. Dieser Angriff erlaubt es Angreifern, durch das automatische Zusenden von Session Cookies und Authentifizierungs-informationen, Aktionen im Namen des angegriffenen Benutzers auszuführen.



Verwundbarkeit

Durch Überprüfung aller Links und Formulare auf unvorhersehbare Token für jeden Benutzer, können Angreifer keine gefälschten Reqüsts unterschieben. Formulare mit zustandsändernden Funktionen sowie mehrstufige Transaktionen sollten gleichermaßen geprüft werden, da diese meist Ziel von CSRF sind. OWASP bietet einen CSRF-Tester an.

Prävention

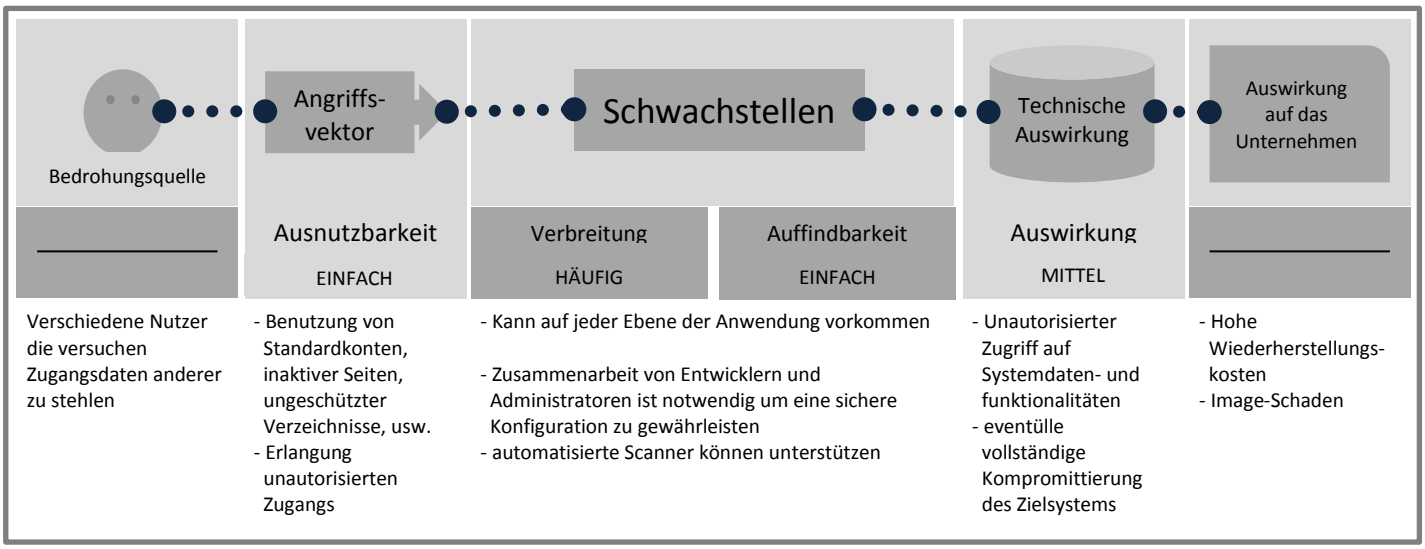
- Unterbringung unvorhersehbarer Token, bestenfalls im Body und für mindestens jede Nutzer-Session, besser jeden Request

Sicherheitsmaßnahme in Django

- Middleware für den Schutz vor CSRF integriert
- Basiert auf Cookies

5.6. Sicherheitsrelevante Fehlkonfiguration

Da viele Konfigurationen für Anwendungen, Frameworks, Applikations-, Web- und Datenbankservern meist nicht mit sicheren Grundeinstellungen geliefert werden, würde eine unregelmäßige Wartung, Definierung, Umsetzung und Aktualisierung zu Schwachstellen führen.



Verwundbarkeit

Konfigurationen sollten Sicherheitshärtungsmaßnahmen auf allen Ebenen verwenden, Patch-Managementprozesse sollten existieren, alle nicht benötigten Komponenten deaktiviert oder entfern werden, Standardkonten deaktiviert oder entfernt werden, Ausgabe von Stack Traces oder technischen Fehlermeldungen verhindert werden und Sicherheitseinstellungen der verwendeten Frameworks sowie verwendeter Bibliotheken verstanden und richtig konfiguriert sein.

Prävention

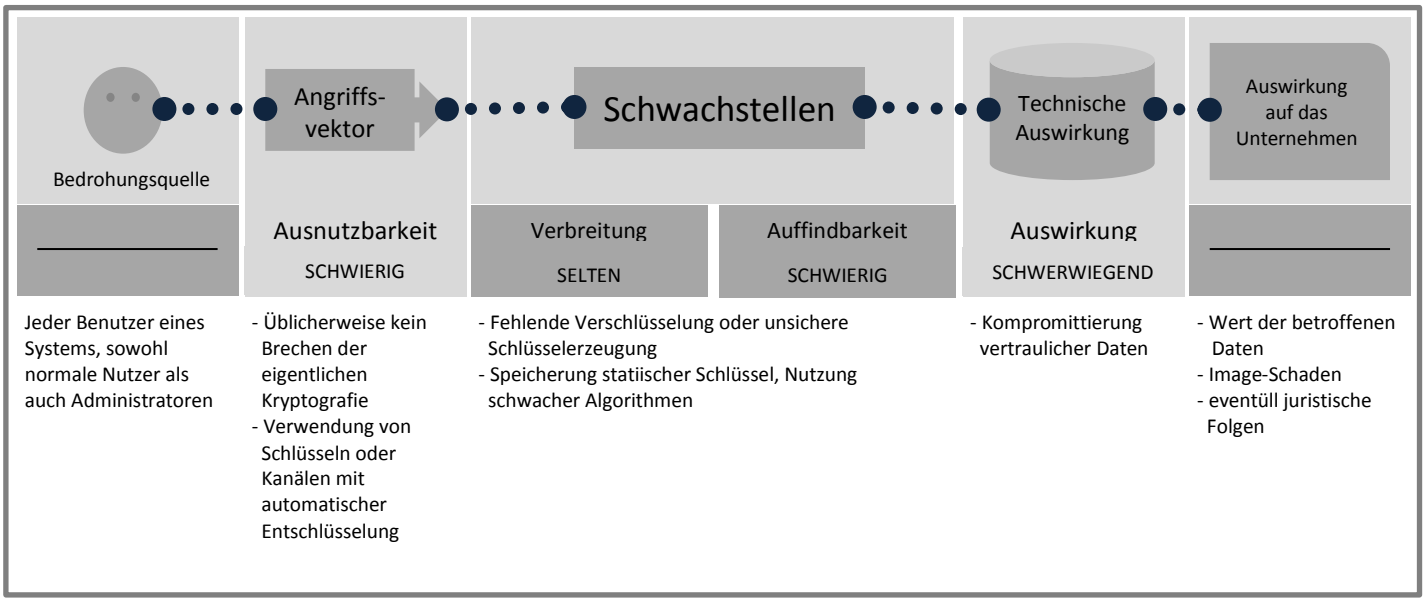
- wiederholbarer Härtungsprozess mit zeitnah neuentwickelten Softwareupdates und Patches für gesamte Umgebung incl. Bibliotheken und Komponenten
- periodisch durchgeführte Tests und Audits

Sicherheitsmaßnahme in Django

- Fehlkonfigurationen liegen nicht in Djangos Verantwortung

5.7. Kryptografisch unsichere Speicherung

Viele sensible Daten werden nicht genug durch Verschlüsselung oder Hashing geschützt, sodass Angreifer diese auslesen und weiterverwenden können.



Verwundbarkeit

Vertrauliche Daten wie Passwörter, Kreditkarteninformationen und personenbezogene Daten sollten auch bei Langzeitspeicherung und Backups per starken, standardisierten Verschlüsselungs-algorithmen geschützt werden. Regelmäßige Schlüsselwechsel sollten stattfinden.

Prävention

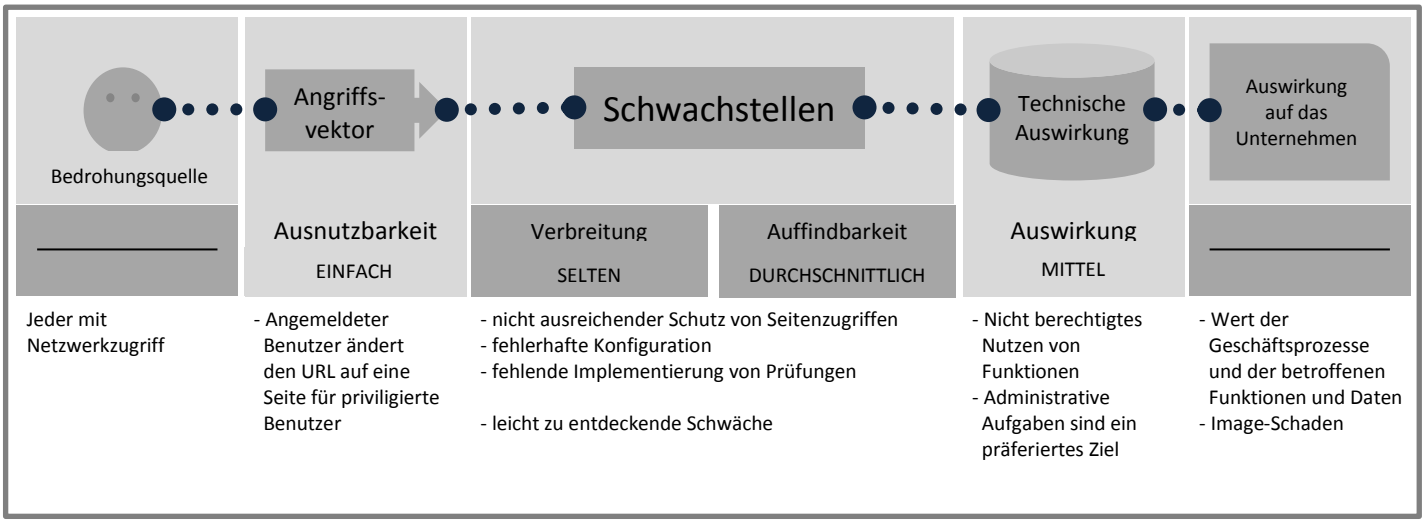
- angemessene Datenverschlüsselung
- getrennte Schlüsselverwaltung
- starken Algorithmus bei Passwörtern sicherstellen

Sicherheitsmaßnahme in Django

- Mit der Standard-Authentifizierung von Django werden Passwörter automatisch mit SHA-1 gehashed
- Weitere Daten müssen außerdem explizit benannt werden

5.8. Mangelhafter URL-Zugriffsschutz

Da viele Anwendungen Zugriffsberechtigungen nur durch das Anzeige oder Ausblenden von Links oder Buttons schützen, können Angreifer durch gezieltes manipulieren von URLs auf diese zugreifen, falls nicht zusätzlich noch eine Prüfung der Zugriffsberechtigung stattfindet.



Verwundbarkeit

Auf jeder Seite ist der Zugriffsschutz zu prüfen, ob öffentlicher oder privater Zugriff erwünscht ist, und ob jeder angemeldete Benutzer Zugriff auf diese hat oder nur nach Berechtigungsprüfung. Externe Sicherheitsmechanismen sollten für jede Seite konfiguriert sein, Schutz auf Code-basis sollte für alle Seiten aktiv sein, Penetrationstests können prüfen, ob geeignete Schutzmechanismen verwendet werden.

Prävention

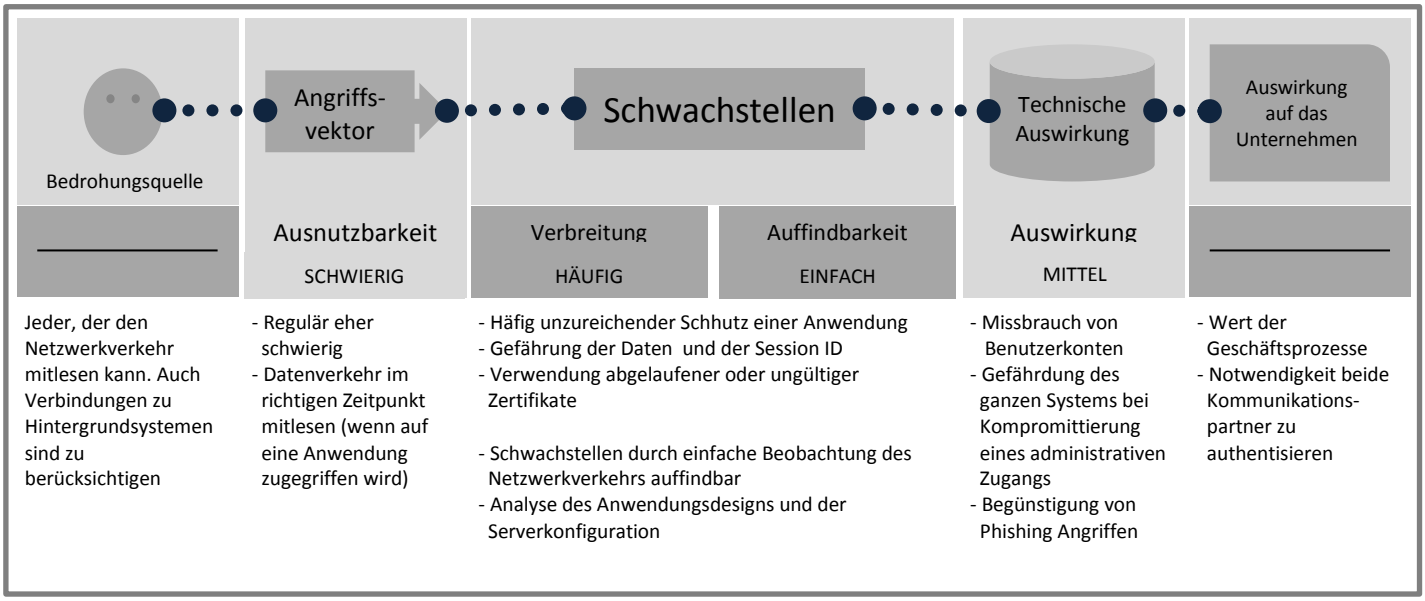
- korrekte Authentifizierung und Autorisierung sollten für jede Seite rollenbasiert und als Richtlinie hinterlegt sein
- Mechanismen sollten in der Standardeinstellung jeglichen Zugriff untersagen

Sicherheitsmaßnahme in Django

- Views können mit spezifischen Rechten/ Rollen ausgestattet werden
- Dies muss jedoch manüll geschehen

5.9. Unzureichende Absicherung der Transportschicht

Viele Anwendung schützen sensiblen Datenverkehr lediglich durch schwache Algorithmen, oder durch abgelaufene oder ungültige Zertifikate. Durch die unzureichende Verschlüsselung können Daten ausgelesen werden.



Verwundbarkeit

Zur Transportschichtsabsicherung sollte SSL zur Absicherung des Netzwerkverkehrs, der mit der Authentifizierung in Verbindung steht, sowie zum Schutz übertragener Daten und Session Tokens für alle Ressourcen auf geschützten Seiten und Diensten, genutzt werden. Verschlüsselte und umverschlüsselte Inhalte sollten zur Vermeidung von Browser Fehlermeldungen und somit zum Schutz der Session ID nicht parallel verwendet werden. Die Benutzung ausschließlich anerkannte Zertifikate ist ebenfalls von Nöten.

Prävention

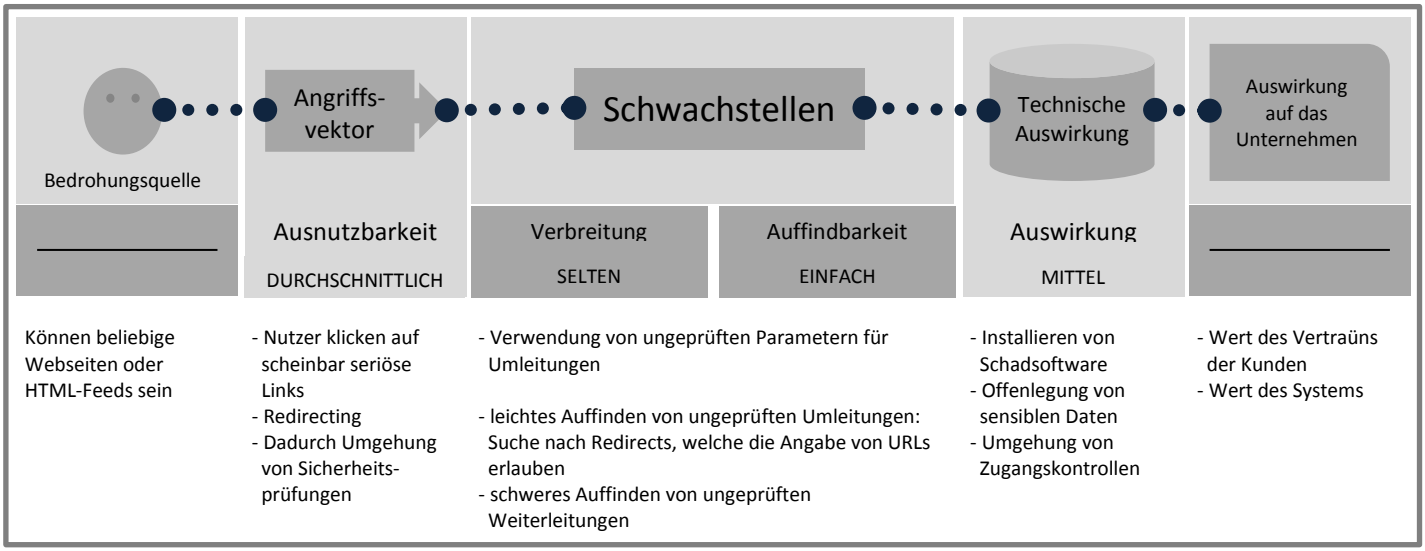
- SSL. nur mit starken Verschlüsselungsalgorithmen auf allen Seiten mit sensiblem Inhalt aktivieren
- wichtige Cookies nur mit Secure-Flag
- nur gültige, nicht abgelaufene und nicht zurückgezogene Zertifikate nutzen

Sicherheitsmaßnahme in Django

- Der Aufbau von sicheren Verbindungen gehört nicht in das Aufgabenfeld von Django
- Jedoch wird ein sogenanntes Secure Cookie Protocol unterstützt

5.10. Ungeprüfte Um- und Weiterleitung

Ohne entsprechenden Schutz können Angreifer andere Benutzer auf Phishing-Seiten oder Seiten mit Schadcode um- oder weiterleiten.



Verwundbarkeit

Eine Code-Prüfung auf Um- und Weiterleitungen mit Beschränkung auf parametrisierte Ziel-URLs genügt als Prüfung.

Prävention

- Weiter- und Umleitungen sowie Zielparameter wenn möglich vermeiden

Sicherheitsmaßnahme in Django

- Django aktiviert automatisch das Modul contrib.auth
- Dadurch sind alle Redirects unzulässig, welche auf einen anderen Host zeigen

6. Fazit

Während die Risiken A1 - Injection, A2 - Cross-Site Scripting, A5 - Cross-Site Reqüst Forgery, A7 - Kryptographisch unsichere Speicherung sowie A10 - Ungeprüfte Um- und Weiterleitungen von Django's Funktionen übernommen werden können, sind Risiken A6 - Sicherheitsrelevante Fehlkonfiguration und A9 - Unzureichende Absicherung der Transportschicht nicht von Django übernehmbar. Dies bedeutet dass Fehlkonfigurationen, z.B. bei A6 nicht in der Verantwortung von Django liegen. Bei den Risiken A3 - Fehler in Authentifizierung und Session Management, A4 - Unsichere direkte Objektreferenzen und A8 - Mangelhafter URL-Zugriffsschutz, sind die Entwickler verantwortlich, sodass beispielsweise für „Unsichere direkte Objektreferenzen" die Zugriffserlaubnis für spezielle Objekte manüll überprüft werden müssen.

N°	Bezeichnung	Einschätzung
1	Injection	
2	Cross-Site Scripting (XSS)	
3	Fehler in Authentifizierung und Session Management	
4	Unsichere direkte Objektreferenzen	
5	Cross-Site Request Forgery (CSRF)	
6	Sicherheitsrelevante Fehlkonfiguration	
7	Kryptografisch unsichere Speicherung	
8	Mangelhafter URL-Zugriffsschutz	
9	Unzureichende Absicherung der Transportschicht	
10	Ungeprüfte Um- und Weiterleitungen	

= Django übernimmt! = Entwickler ist verantwortlich!
= Django übernimmt nicht!

Abbildung 3

Anhand der farblich unterlegten Tabelle des Testing Guides sieht man, dass Django im Bereich Session Management alle zu testenden Schutzfunktionen übernimmt, während es sich zum Beispiel für das Schützen der Informationssuche und des Konfigurationsmanagements nicht eignet.
Somit bietet Django auf der einen Seite viele relevante Sicherheitsmechanismen, andererseits liegt es an den Fähigkeiten der Entwickler die Tools richtig zu implementieren um eine Anwendung sicherer zu machen.

Information Gathering	Configuration Management Testing	Authentication Testing	Session Management	Authorization Testing	Business logic testing	Data Validation Testing	Denial of Service Testing	Web Services Testing	AJAX Testing
Spiders, Robots and Crawlers	SSL/TLS Testing	Credentials transport over an encrypted channel	Testing for Session Management Schema	Testing for Path Traversal	Testing for business logic	Testing for Reflected Cross Site Scripting	Testing for SQL Wildcard Attacks	WS Information Gathering	AJAX Vulnerabilities
Search Engine Discovery	DB Listener Testing	Testing for user enumeration	Testing for Cookies attributes	Bypassing authorization schema		Testing for Stored Cross Site Scripting	Locking Customer Accounts	Testing WSDL	AJAX Testing
Identify application entry points	Infrastructure Configuration Management Testing	Testing for Guessable (Dictionary) User Account	Testing for Session Fixation	Testing for Privilege Escalation		Testing for DOM based Cross Site Scripting	Testing for DoS Buffer Overflows	XML Structural Testing	
Testing for Web Application Fingerprint	Application Configuration Management Testing	Brute Force Testing	Testing for Exposed Session Variables			Testing for Cross Site Flashing	User Specified Object Allocation	XML content-level Testing	
Application Discovery	Testing for File Extensions Handling	Testing for bypassing authentication schema	Testing for CSRF			SQL, LDAP, ORM, XML, SSI, XPATH, SMTP, Code Injection	User Input as a Loop Counter	HTTP GET parameters/ REST Testing	
Analysis of Error Codes	Old, backup and unreferenced files	Testing for vulnerable password and reset				OS Commanding	Writing Data to Disk	Naughty SOAP attachments	
	Admin Interfaces	Logout and Browser Cache Management				Buffer overflow	Failure to Release Resources	Replay Testing	
	Testing for HTTP Methods and XST	Testing for CAPTCHA				Incubated vulnerability Testing	Too Much Data in Session		
		Testing Multiple Factors Authentication				Testing for HTTP Splitting/ Smuggling			
		Testing for Race Conditions							

Abbildung 4

Abbildungsverzeichnis

Referenzen

https://docs.djangoproject.com/en/dev/topics/security/

https://www.owasp.org/images/b/b8/OWASPTop10_DE_Version_1_0.pdf

http://www.panisch.com/2012/02/sicherheitsanalyse-des-django-webframeworks/