# IWS-Workshop: KI oder PR1 im Studium:

# Wie viele Schleifenarten muss man kennen?

## Lösung Aufgabe: Ethics

## Schwachstellenthema 1: Buffer-Overflow

### Code Snippet 1:

| Zu identifizierende Schwachstelle | Erklärung |
|---|---|
| 1. Ungeprüfte Nutzereingabe<br>2. L.7: Speicher-überschreibung | 1. Die gets() Funktion wird für Nutzereingaben verwendet, jedoch ohne Bound Checkings. Dies kann zu einem Buffer-Overflow führen<br>2. Nachdem die Grenze des Buffers überschritten wurde, kann der Angreifer nun beliebigen (schädlichen) Code in den anliegenden Speicher schreiben |
| Auswirkungen:<br>• Programm Absturz<br>• Ausführung von beliebigem (schädlichen/nicht beabsichtigter) Code<br>• Unvorhersehbares Verhalten<br>• Angreifer könnten dadurch: unbefugten Zugang zu Systemen, Vertraulichkeit von Daten gefährden, Daten leaken/einsehen | |

## Schwachstellenthema 2: SQL-Injection

### Code Snippet 1:

| Zu identifizierende Schwachstelle | Erklärung |
|---|---|
| L.11: SQL-Injection | 1. Durch die Nutzereingabe könnte ein Angreifer die Logik des SQL Query manipulieren/modifizieren, um schädliche Handlungen durchzuführen, wie z. B. Sensible Daten auslesen, Daten modifizieren oder löschen, beliebige Befehle auf der Datenbank ausführen<br>2. Der req.body.username wird direkt in die SQL Query zusammengeführt ohne jegliche Überprüfung. Das heißt: Ein Angreifer könnte das Eingabefeld dazu ausnutzen, beliebigen SQL-Code einzuspeisen |
| Auswirkungen: | |

| |
|---|
| • Auslesen von Datenbank Inhalten<br>• Manipulation der Daten (Modifizieren, Löschen, …)<br>• Unbefugter Zugriff  Adminrechte |

### Code Snippet 2:

| Zu identifizierende Schwachstelle | Erklärung |
|---|---|
| 1. L.18: Unverschlüsseltes Password<br>2. L.5/21: Kein Error Handling<br>3. L.9/L.25: Nutzung von Variable valTom ohne Validierung | 1. Das Speichern des Passworts direkt in die Datenbank ist unsicher. Sollte ein Angreifer Zugriff auf die Codebasis bekommen, kann das Passwort einfach erlangt werden.<br>2. Kein Error Handling, wenn die http Request fehlschlägt. Sollte der Fehler nicht abgefangen werden, könnte beispielsweise sensitive Daten über http ausgegeben werden<br>3. ValTom koennte Nutzerinput sein, welches nicht überprüft wird. Dies könnte zu einer SQL Injection führen |
| Auswirkungen:<br>• Auslesen von Datenbank Inhalten<br>• Manipulation der Daten (Modifizieren, Löschen, …)<br>• Unbefugter Zugriff  Adminrechte | |

### Code Snippet 3:

| Zu identifizierende Schwachstelle | Erklärung |
|---|---|
| L.7 | Die Nutzereingabe wird in die Query ohne weiteres zusammengeführt, anstatt als nicht vertrauenswürdige Daten behandelt zu werden |
| Auswirkungen:<br>• Auslesen von Datenbank Inhalten<br>• Manipulation der Daten (Modifizieren, Löschen, …)<br>• Unbefugter Zugriff  Adminrechte | |

## Schwachstellenthema 3: SSRF

### Code Snippet 1:

| Zu identifizierende Schwachstelle | Erklärung |
|---|---|
| L.11: | 1. Die Funktion downloadURL nimmt einen URL-Parameter (url) entgegen und stellt eine GET-Anforderung an diese URL unter |

| | |
|---|---|
| 1. Keine Eingabeüberprüfung der URL-Parameter<br>L.18:<br>2. Keine Begrenzung an Anfragen | Verwendung der 'request'-Bibliothek. Der URL-Parameter wird ohne Validierung oder Einschränkung direkt aus dem Anfragetext übernommen.<br>2. Anfälligkeit für DoS-Attacken. Ein Angreifer könnte den Server mit Anfragen überschwemmen, seine Ressourcen aufbrauchen und letztendlich verlangsamen bis lahmlegen |

# Schwachstellenthema 4: XSS

| Zu identifizierende Schwachstelle + Erklärung |
|---|
| L.15: Nimmt Argumente an, ohne die Eingabe zu überprüfen<br><br>L.15: async void (Kein Rückgabewert), ist nicht empfohlen (in ASP.NET Anwendungen), kann Probleme mit Ausnahme Handling führen<br><br>L.19: Der UserInfo Parameter wird nicht überprüft, bevor es in die Response geschrieben wird. Dadurch könnte potenziell schädlicher Code injecten werden. |
| Auswirkungen:<br>• Ausführen von schädlichen Skripten, wenn ein Nutzer auf die Webseite geht<br>• Klauen von sensiblen Daten / Datendiebstahl<br>• Verbreitung auf weitere Nutzeraccounts: Potenziell könnten weitere Nutzer (die auf die Webseite draufgehen und das Skript ausgeführt wird) davon betroffen werden und den Angriff eines Angreifers vervielfachen |