

DOKUMENTATION

Broken Web Application

Ein Einstieg in die Sicherheit von Web-Applikationen

Master Informatik

Studentisches Projekt

Betreuer *Prof. Dr. Stefan Hahndel*
Prof. Dr. Ernst Göldner

Wintersemester 2017/18



Technische Hochschule
Ingolstadt

Inhaltsverzeichnis

Abbildungsverzeichnis	4
1 Einleitung	5
1.1 Abschnitt 1	5
1.2 Abschnitt 2	5
2 Fachbegriffe	6
2.1 Fachbegriff 1	6
2.2 Fachbegriff 2	6
3 Vorbereitung	7
3.1 Installationsanleitung	7
3.2 Nutzung der virtuellen Maschine	8
4 Verwendete Tools	9
4.1 Tool 1	9
5 Broken Authentication and Session Management	10
5.1 Erklärung	10
6 Buffer Overflow	11
6.1 Erklärung	11
7 Cross-Site-Scripting	12
7.1 Erklärung	12
7.1.1 Angriffsvarianten	12
7.2 Ablauf	13
7.2.1 Reflected Cross-Site-Scripting	13
7.2.2 Stored Cross-Site-Scripting	13
7.2.3 DOM-Based Cross-Site-Scripting	15
8 Login Parcours	17
8.1 Erklärung	17
8.2 Ablauf	17
8.2.1 Level 1 - JavaScript-Code	18

8.2.2	Level 2 - Decoding JavaScript-Code	20
8.2.3	Level 3 - Source Code Suche	20
8.2.4	Level 4 - Versteckte Dateien	21
8.2.5	Level 5 - Caesar-Verschlüsselung	22
8.2.6	Level 6 - Wireshark-Sniffing	25
8.2.7	Level 7 - BCrypt-Verschlüsselung	26
8.3	Ausblick	27
9	SQL-Injection	29
9.1	Erklärung	29
9.1.1	Grundlagen Datenbanksysteme	29
9.1.2	3-Schichten-Architektur	30
9.1.3	Der Angriff	31
9.2	Vorbereitung	31
9.3	Ablauf	31
9.3.1	Aufbau des Login-Web-Services	31
9.3.2	SQL-Injection zum Auslesen von Daten	33
9.3.3	SQL-Injection zum Einfügen von Daten	35
9.3.4	SQL-Injection zum Löschen von Tabellen	36
9.3.5	SQL-Injection zum Modifizieren von Tabellen	37
9.3.6	Die SQL-Injection-Spielwiese	38
9.4	Gegenmaßnahmen	39
9.4.1	Prepared Statements	39
9.4.2	Escapen von Eingaben	40
10	Disclaimer	41

Abbildungsverzeichnis

7.1	Darstellung des theoretischen Hintergrund für Stored-XSS	14
7.2	Darstellung des theoretischen Hintergrund für Stored-XSS	15
7.3	Aufgabenstellung des DOM-based-XSS Tutorials	16
8.1	Überblicksseite des Login Parcours	18
8.2	Login-Maske des ersten Levels	19
8.3	Lösung des ersten Levels	20
8.4	Codeausschnitt zur Lösung des zweiten Levels	20
8.5	Codeausschnitt zur Lösung des dritten Levels	21
8.6	Gefundene Datei des vierten Levels	21
8.7	Lösung des vierten Levels	22
8.8	Umsetzungstabelle bei der Caesar-Verschlüsselung mit Shift 3	23
8.9	Login-Maske der Caesar-Verschlüsselung im fünften Level	24
8.10	Login-Maske des sechsten Levels	25
8.11	Wiresharkausschnitt für die Lösung des sechsten Levels	25
8.12	BCrypt Login Maske mit gegebenen Credentials	27
9.1	3-Schichten-Architektur	30
9.2	Tabellenstruktur der Tabelle „secretUserData“	32
9.3	Login-Oberfläche des Web-Services	32
9.4	Normaler Login	33
9.5	Login mit SELECT-Injection	34
9.6	Auswertung von OR "1"=1	34
9.7	TODO: aktuelles Bild Login mit INSERT-Injection	35
9.8	TODO: Bild aktualisieren Login mit DROP-Injection	36
9.9	Rangliste mit Suchfunktion	37
9.10	Rangliste mit gesuchtem User	38
9.11	Rangliste mit gesuchtem User	38
9.12	TODO: aktuelles bild Verschiedene SQL-Injections zum Ausprobieren	39

1 Einleitung

Einleitung ...

1.1 Abschnitt 1

Erster Abschnitt

1.2 Abschnitt 2

Zweiter Abschnitt

2 Fachbegriffe

Hier werden wichtige Fachbegriffe im Kontext der Broken Web Applikation kurz erklärt, die im späteren Verlauf für die einzelnen Aufgaben eine Rolle spielen.

2.1 Fachbegriff 1

Das ist der erste Fachbegriff.

2.2 Fachbegriff 2

Das ist der zweite Fachbegriff.

3 Vorbereitung

In diesem Kapitel soll erläutert werden, wie das Web-Projekt bereitgestellt und benutzt wird. Hierzu folgt eine kurze Anleitung um einen Host zu installieren, der die Web-Applikation bereitstellt sowie eine Beschreibung mit dem Umgang der bereits installierten virtuellen Maschine.

3.1 Installationsanleitung

Für die Installation der Web-Applikation muss zuerst ein Webserver auf dem Linux Host-Gerät installiert werden, wie z. B. der Apache Server (<https://httpd.apache.org>). Eine einfach Installation ist mit dem Befehl `apt-get install apache2` möglich. Da sich das Installations-Vorgehen mit den Versionen der Drittanbieter-Software ändert, verzichte ich hier auf eine detaillierte Schritt-für-Schritt-Anleitung. Damit das Webprojekt richtig ausgeführt wird, muss in der `apache2.conf` folgende zwei Zeilen eingefügt werden:

- `RemoveHandler .html .htm`
- `AddType application/x-httpd-php .php .htm .html`

Diese Konfiguration sorgt dafür, dass HTML-Seiten mit dem PHP-Interpreter ausgeführt werden. Nach der Konfiguration des Apache-Webservers, muss der PHP-Interpreter installiert werden, dies kann ebenfalls mit dem Package-Manager durchgeführt werden. Dazu wird das Kommando `apt-get install php` verwendet. Zum Zeitpunkt der Entwicklung des Projekts, wurde die PHP-Version 7.0.19 verwendet. Um die Installation auf Korrektheit zu prüfen, kann der Befehl `php --version` benutzt werden. Dieser zeigt die Versionsnummer der PHP-Umgebung, falls die Installation korrekt durchgeführt wurde.

Um alle Tutorials durchführen zu können muss der Host einen GNU-Debugger (GDB) bereitstellen. Dieser kann leicht mit `apt-get install gdb` bezogen werden. Um den GDB zu testen, kann der Befehl `gdb --version` ausgeführt und die Version angezeigt werden.

Zuletzt muss noch eine MySQL-Datenbank auf dem Host-System installiert werden. Hierfür stehen mehrere Alternative Vorgehensweisen zur Verfügung, siehe dazu <https://dev.mysql.com/doc/refman/5.7/en/linux-installation.html>.

Um nun die Applikation verwenden zu können, müssen zunächst alle Web-Ressourcen aus dem GitHub-Repository in den von Apache erwarteten Pfad kopieren. Dazu muss das Repository auf den Server kopiert werden, dies kann mit dem Befehl `git clone https://github.com/th-ingolstadt/INF-M-Projekt-Security-Workbench.git` erreicht werden. Danach muss der Inhalt des Pfads `PROJEKTROOT/Projekte/-SecWorkbench/html` in das Root-Verzeichnis des Apache-Servers kopiert werden. Dieser ist standardmäßig `/var/www/html`. Des Weiteren müssen die Rechte des Apache-Users auf das Verzeichnis angepasst werden, dies ist mit dem folgenden Befehl möglich `xxxxxx`.

3.2 Nutzung der virtuellen Maschine

4 Verwendete Tools

Es folgt eine kurze Übersicht der Tools, die in den Beispielen mehrfach eingesetzt werden. Hier wird jeweils der Zweck des Tools und die Bedienung kurz demonstriert.

4.1 Tool 1

5 Broken Authentication and Session Management

Broken Authentication and Session Management...

5.1 Erklärung

Erklärung...

6 Buffer Overflow

Buffer Overflow ...

6.1 Erklärung

Erklärung ...

7 Cross-Site-Scripting

Cross Site Scripting

7.1 Erklärung

blabla bla bla

7.1.1 Angriffsvarianten

Grundlegend gibt es drei verschiedene Arten von Cross-Site-Scripting Angriffen: Reflected XSS, Stored XSS, DOM-Based XSS . Um diese möglichst einfach zu erklären, wird im Folgenden der JavaScript Code `alert("42")` als Beispiel verwendet. Hierbei kann aber auch jeder andere beliebige JavaScript Code eingeschleust werden. Je nachdem was das Ziel des Angreifers ist, werden bspw. mit `alert(document.cookie)` , die gespeicherten Cookies der Webseite auslesen.

Reflected XSS

Stored XSS

Im Vergleich zum reflektierenden XSS-Angriff unterscheidet sich der Stored XSS (auch persistent/ persistentes XSS) dadurch, dass der Schadcode auf dem Webserver gespeichert wird. Besonders problematisch dabei ist, dass bei jeder Clientanfrage der Schadcode automatisch ausgeliefert und ausgeführt wird.

DOM-Based XSS

Die dritte Angriffsart von Cross-Site-Scripting bezieht sich ausschließlich auf statische HTML-Seite mit JavaScript Unterstützung lässt dabei den Server außen vor.

7.1.1.1 Gegenmaßnahmen

...

Abschließend muss erwähnt werden, dass bestimmte Browser wie Google Chrome bereits einen integrierten Cross-Site-Scripting Schutz bieten. Daher ist es zwingend notwendig, dass das nachfolgende Tutorial unter einem Browser ohne XSS-Schutz (z. B. Firefox) durchgeführt wird.

7.2 Ablauf

Das nachfolgende Kapitel beschreibt den Ablauf der Cross-Site-Scripting Tutorials. Hierzu werden zuerst auf die verschiedenen Übungen des reflektierenden Cross-Site-Scripting vorgestellt. Daraufhin werden jeweils die beiden Aufgaben zu Stored-XSS und DOM-based-XSS beschrieben. Des Weiteren wird zu jeder Beispielaufgabe der Lösungsweg dargelegt.

7.2.1 Reflected Cross-Site-Scripting

7.2.2 Stored Cross-Site-Scripting

Im Tutorial für die Stored-XSS findet der Einsteiger ein Gästebuch wieder, dem Einträge hinzugefügt werden können. Hierzu sind diese mit einer Datenbank gekoppelt und werden dauerhaft gespeichert. Das Laden der Seite führt dazu, dass die Anwendung sich den Inhalt der Datenbank holt und in eine Tabelle schreibt. Um hier eine Sicherheitslücke zu schaffen werden die Usereingaben nicht überprüft und ungefiltert übermittelt.

Die Abbildung 7.1 zeigt das Gästebuch. Folglich soll der Einsteiger hier ein XSS-Angriff durchführen und die Cookies auslesen. In diesen sind die Credentials mit Username/ Passwort fälschlicherweise gespeichert.

Aufgabe

Hinterlasse eine Eintrag in admin's Gästebuch

Submit

admin's Gästebuch

Username	Datum	Eintrag
admin	2017-11-15	Guten Tag! Es freut mich sehr, dass ich hiermit den ersten Beitrag in unserem Gästebuch erstellen darf und möchte dich herzlich Willkommen heißen! :)
admin	2017-11-15	Was wohl mit diesem stored cross site scripting alles möglich ist
admin	2017-11-16	Wenn ich an Cookies denke, dann frage ich mich manchmal was dort drin steht.. manchmal hab ich auch einfach nur Lust auf einen Cookie :D
Mr. Robot	2017-11-18	42
root	2017-11-20	Hello World!
root	2017-11-21	1 + 1 = 0
Mr. Robot	2017-11-22	Knock, knock. Race condition. Who's there.
Mr. Robot	2017-11-25	Why do Java programmers wear glasses? Because they don't C#!

Abbildung 7.1: Darstellung des theoretischen Hintergrund für Stored-XSS

Die Lösung für das Tutorial ist es somit, dass der Einsteiger einen Gästebucheintrag verfasst, in dem ein Skript injiziert wird. Ein Beispiel hierfür ist `<script>alert("Hello World!");</script>`. Da es für die Lösung der Aufgabe erforderlich ist die Cookies auszulesen, muss der Eintrag die folgende Zeichensequenz enthalten:

```
<script>alert("document.cookie");</script>
```

Um den Einsteiger optimal auf die Übung vorzubereiten, besitzt dieses Tutorial ein Video mit dem theoretischen Hintergrund für Stored-XSS. Zudem werden dem Einsteiger Tipps an die Hand gegeben um die Aufgabe zu lösen. Abschließend ist

zu erwähnen, dass auf dieser Tutorialseite auch die Gegenmaßnahmen dargestellt sind. Die Abbildung 7.2 veranschaulicht das eben Beschriebene.

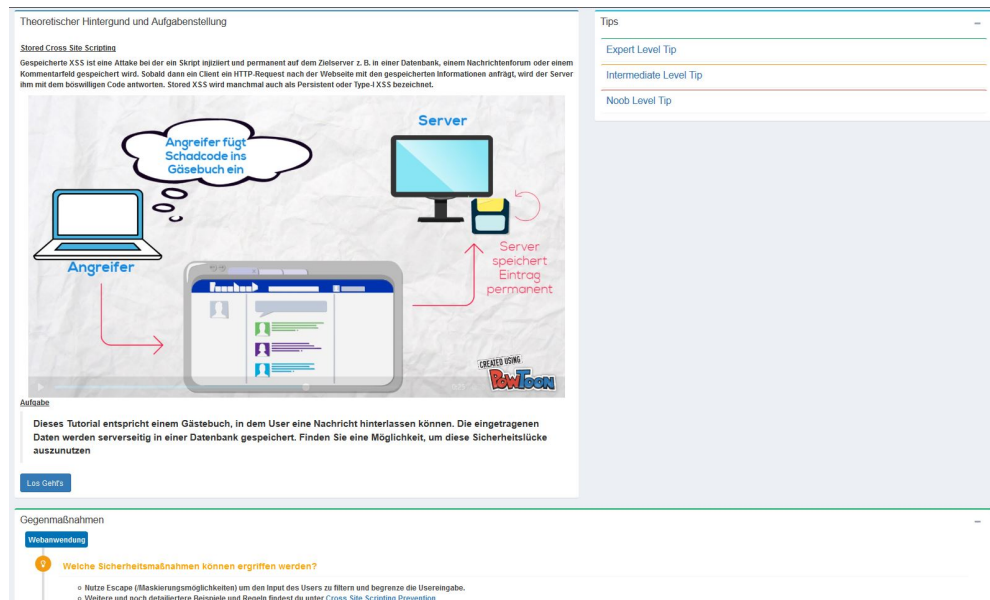


Abbildung 7.2: Darstellung des theoretischen Hintergrund für Stored-XSS

7.2.3 DOM-Based Cross-Site-Scripting

Das letzte Tutorial in Bezug auf XSS-Angriffe befasst sich mit der Variante des DOM-Based-XSS.

Hierzu wurde exemplarisch eine Sprachauswahl implementiert, bei der eine Selektierungsmöglichkeit über die URL erzeugt wird. Das folgende Listing 7.1 zeigt diesen Anwendungskontext:

Listing 7.1: Select-Statement

```
<select id="languageList" style='width:500px'>
  <script>
    document.write("<OPTION value=1>" +
      document.location.href.substring(document.
        location.href.indexOf("default=") + 8) + "</OPTION>");
  </script>
</select>
```

Hier wird eine Select-Option über JavaScript erzeugt, indem auf die URL zugegriffen wird. Die Default URL für dieses Tutorial ist `http://localhost/html/DOMbasedXSS.html?default=german`.

Folglich entspricht eine Auswahlmöglichkeit der Sprache *german*, die restlichen Wahlmöglichkeiten werden über JavaScript inkludiert. Die Abbildung 7.3 zeigt die Aufgabenstellung.

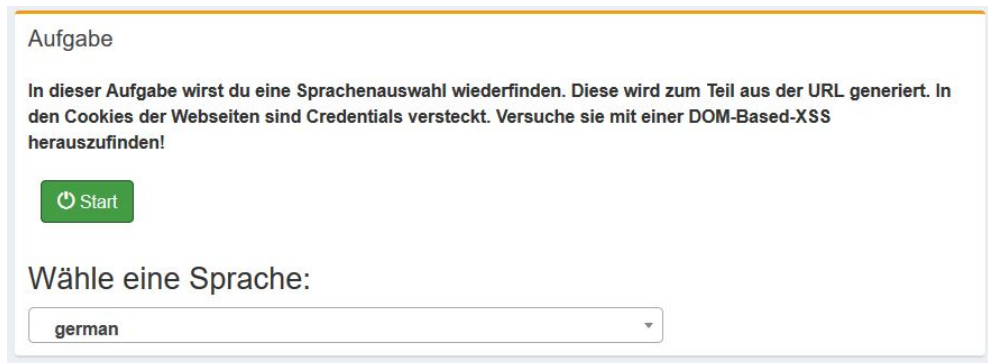
The screenshot shows a web interface for a task titled 'Aufgabe'. The text explains that the user must find a language selection that is generated from the URL and stored in cookies, and that they should attempt to extract it using DOM-based XSS. There is a green 'Start' button with a play icon. Below this, there is a label 'Wähle eine Sprache:' followed by a dropdown menu that currently displays 'german'.

Abbildung 7.3: Aufgabenstellung des DOM-based-XSS Tutorials

Um das Tutorial zu lösen muss der Einsteiger die URL manipulieren, indem er dort ein Skript in die Anwendung injiziert. Hierzu soll der erneut die Cookies auslesen. Die Lösung für das Tutorial ist somit:

```
.../DOMbasedXSS.html?default=german<script>alert("document.cookie");</script>
```

Abschließend muss hier erwähnt werden, dass auch dieses Tutorial ein Theorievideo sowie Tipps und Gegenmaßnahmen besitzt.

8 Login Parcours

Das nachfolgende Kapitel beschreibt den Login Parcours der Broken-Web-Applikation. Hierzu folgt eine kurze Erläuterung um was es sich bei diesem Parcours handelt. Anschließend werden die einzelnen Übungslevel vorgestellt und gezeigt wie diese zu lösen sind. Abgeschlossen wird das Kapitel mit einem kurzen Ausblick wie der Login Parcours erweitert werden kann.

8.1 Erklärung

Der Login Parcours umfasst eine Reihe von Rätseln, bei denen der Einsteiger versuchen muss, das Passwort einer gegebenen Login-Maske herauszufinden. Hierzu enthält jedes Level eine kurze Beschreibung mit Hinweisen, wie das Rätsel zu lösen ist (z. B. in dem der JavaScript-Code inspiziert oder eine unverschlüsselte PCAP Datei ausgelesen werden soll).

Das Ziel des Parcours ist es das Interesse des Einsteigers zu wecken, indem er einfache Aufgabestellungen lösen muss, die dem Finden von Sicherheitslücken ähneln. Zudem soll die Denkweise in Bezug auf diese Lücken geschult und gängige Gegenstände der Themengebiete wie Verschlüsselung näher gebracht werden. Darüber hinaus soll hierbei ein Spaßfaktor entstehen, sodass die Einsteiger Lust bekommen sich genauer mit der Thematik der IT-Sicherheit zu beschäftigen.

8.2 Ablauf

Der Einstiegspunkt zu den Aufgaben entspricht der Überblickseite (Siehe Abbildung 8.1) des Login-Parcours. Dort ist jedes Level mit kurzer Überschrift bzgl. der Art des Rätsels verlinkt.

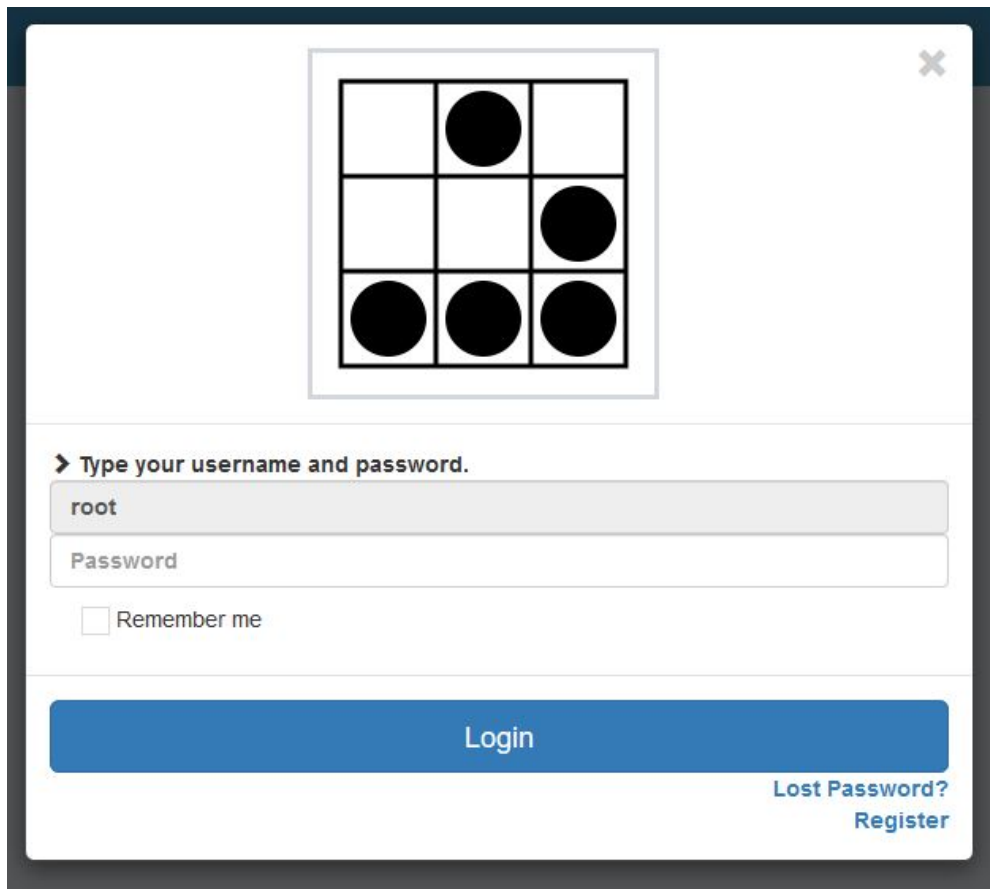


Abbildung 8.1: Überblicksseite des Login Parcours

Der prinzipielle Ablauf jedes Rätsels ist gleich. Der Anwender bekommt einen Startbutton, mit dem eine Login-Maske erscheint. Diese ist mit einer Kurzbeschreibung ausgestattet. Wird ein Rätsel gelöst, so erscheint die dazugehörige Erfolgsmeldung und ein Verweis auf das nächste Level.

8.2.1 Level 1 - JavaScript-Code

Im ersten Level muss der Einsteiger den JavaScript-Code auslesen, in dem fälschlicherweise die Validierung des Usernamen und Passworts enthalten ist. Der Start des Levels entspricht der Login-Maske wie in Abbildung 8.2.



The image shows a login interface for a game level. At the top, there is a 3x3 grid of squares. The top row has a black circle in the middle square. The middle row has a black circle in the right square. The bottom row has black circles in all three squares. Below the grid, there is a prompt: 'Type your username and password.' followed by two input fields. The first field contains the text 'root'. The second field is labeled 'Password'. Below these fields is a checkbox labeled 'Remember me'. At the bottom, there is a large blue button labeled 'Login'. To the right of the button, there are two links: 'Lost Password?' and 'Register'.

	●	
		●
●	●	●

➤ Type your username and password.

root

Password

☐ Remember me

Login

[Lost Password?](#)
[Register](#)

Abbildung 8.2: Login-Maske des ersten Levels

Der Einsteiger soll sich mit einem Rechtsklick und `Seitenquelltext anzeigen`, den Sourcecode anzeigen lassen. Wird dieser genauer betrachtet so wird erstichtlich, dass die Validierung des Usernamen und Passwort in einem Skript geschieht. Dort findet der Einsteiger auch das Passwort `hallo123` zum vorgegebenen Usernamen `root`. Abbildung 8.3 zeigt den zugehörigen Codeausschnitt.

```

<script>
function checkLoginCredentials() {
    var password = document.getElementById('login_password');
    if (password.value == "hallo123") {
        $('#login-modal').modal('hide');
        document.getElementById("solution").removeAttribute("hidden");
        document.getElementById("instruction").setAttribute("hidden", true);
    } else {
        alert("Wrong password - try again!");
    }
}
}
</script>

```

Abbildung 8.3: Lösung des ersten Levels

8.2.2 Level 2 - Decoding JavaScript-Code

Das zweite Level ähnelt prinzipiell dem Ersten, bei dem der Einsteiger erneut den JavaScript-Code inspizieren soll. Bei genauerer Betrachtung wird hierbei ersichtlich, dass das Passwort mit der JavaScript Funktion `String.fromCharCode(...)` verschlüsselt ist. Diese interpretieren Integerzahlen als Werte des ASCII-Zahlensatz. Abbildung 8.4 zeigt den relevanten Codeausschnitt.

```

<script src="../../Content/js/loginParcour.js"></script>

<script>
function checkLoginCredentials() {
    var password = document.getElementById('login_password');
    if (password.value === String.fromCharCode(112, 97, 115, 115, 119, 111, 114, 100, 49)) {
        $('#login-modal').modal('hide');
        document.getElementById("solution").removeAttribute("hidden");
        document.getElementById("instruction").setAttribute("hidden", true);
    } else {
        alert("Wrong password - try again!");
    }
}
}
</script>

```

Abbildung 8.4: Codeausschnitt zur Lösung des zweiten Levels

Erkenntlich wird dabei, dass das Passwort aus der Zahlenfolge `112, 97, 115, 115, 119, 111, 114, 100, 49` besteht. Übersetzt man diese nun in eine ASCII-Zeichenfolge entsteht das Wort `password1`, dass die Lösung dieses Rätsels entspricht.

8.2.3 Level 3 - Source Code Suche

Im folgenden Level muss der Anwender den Source Code zuerst finden, bevor er die Lösung herauslesen kann. Hierbei wurde in diesem Level der Rechtsklick mit der Maus deaktiviert. In der Praxis ist es jedoch nicht möglich den Source Code

zu verstecken, da dieser auf der Clientseite und somit im Browser vorliegen muss. Der Einsteiger muss folglich auf die Idee kommen, den Source Code mit der richtigen URL-Anfrage auszulesen. Die Lösung hierzu ist die Anweisung `view-source:http://`. Die Abbildung 8.5 zeigt wiederum den relevanten Codeausschnitt mit dem Passwort.

```
<script>
function checkLoginCredentials() {
    var password = document.getElementById('login_password');
    if (password.value === String.fromCharCode(100, 101, 98, 117, 103, 103, 101, 114, 70, 111, 117, 110, 100)) {
        $('#login-modal').modal('hide');
        document.getElementById("solution").removeAttribute("hidden");
        document.getElementById("instruction").setAttribute("hidden", true);
    } else {
        alert("Wrong password - try again!");
    }
}
}
</script>
```

Abbildung 8.5: Codeausschnitt zur Lösung des dritten Levels

Auch hier wurde eine Zahlenfolge gewählt, die in ASCII-Zeichen umgewandelt wird. So generiert sich das Passwort für dies Level aus der Zahlenfolge 100, 101, 98, 117, 103, 103, 101, 114, 70, 111, 117, 110, 100 und entspricht `debuggerFound`.

8.2.4 Level 4 - Versteckte Dateien

Im vierten Level ist eine Datei versteckt, die dem Einsteiger Hinweise zur Lösung des Rätsels gibt. Die Beschreibung des Rätsels ist wie folgt:

Bob ist kein guter Entwickler. Er hat das Passwort vergessen und murmelt die ganze Zeit nur etwas von aliens.txt. Ich weiß auch nicht was das bedeuten soll?!

Um das Rätsel zu lösen muss der Einsteiger, die Datei `aliens.txt` in der URL suchen. Abbildung 8.6 zeigt die gefundene Textdatei.



Abbildung 8.6: Gefundene Datei des vierten Levels

In dieser Textdatei wird auf eine versteckte Seite verwiesen, die in der gesamten Webanwendung durch keinen Link erreichbar ist. Die Abbildung 8.7 zeigt die versteckte Seite auf der das Passwort `bobIsSoStupid123` ersichtlich ist.



Abbildung 8.7: Lösung des vierten Levels

Abschließend muss in diesem Level erwähnt werden, dass die Validierung des Passworts nicht browserseitig geschieht, sondern mittels PHP auf dem Server. Folglich ist es dem Einsteiger nicht wie bisher möglich, das Passwort im JavaScript Code wiederzufinden.

8.2.5 Level 5 - Caesar-Verschlüsselung

Das fünfte Level umfasst eine Caesar-Verschlüsselung. Um folglich das Level zu lösen muss der Einsteiger wissen worum es sich bei dieser Verschlüsselung handelt.

Die Caesar-Verschlüsselung

Gerade die Caesar-Verschlüsselung bietet sich als Einstiegs- und Demonstrationsverschlüsselung an, da mit ihr leicht die Grundlagen der Kryptographie gezeigt werden können. Allerdings ist das Verwenden der Caesar-Verschlüsselung nicht sehr sicher und kann leicht geknackt werden. Dieses Level des Login Parcours soll dies demonstrieren.

Prinzipiell ist die Caesar-Verschlüsselung ein einfaches symmetrisches Verschlüsselungsverfahren aus Zeiten der Römer und geht auf den Feldherr Julius Caesar zurück.

Konkret basiert die Caesar Chiffre auf einer monographischen und monoalphabetischen Substitution. Ausgangspunkt ist das Alphabet so wie wir es kennen. Nun wird jeder Buchstaben durch einen neuen Buchstaben ersetzt, nämlich der Buchstabe, der um drei Stellen versetzt ist. Aus einem Ä wird durch eine Verschiebung um drei Zeichen also der Buchstabe "D", ein "B" wird ein "E", das "C" ein "F" usw. Da das ganze zyklisch ist, wird das "X" durch ein "A" ersetzt, das "Y" durch ein "B" und das "Z" durch ein "C". Damit ergibt sich am Ende folgende Umwandlungstabelle (Siehe Abbildung 8.8):

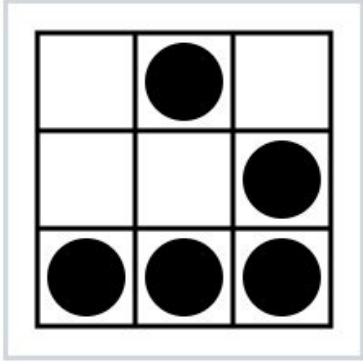
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Abbildung 8.8: Umsetzungstabelle bei der Caesar-Verschlüsselung mit Shift 3

Soll nun bspw. das Wort *security* verschlüsseln, muss lediglich die obige Tabelle nach dem Buchstaben durchsucht werden und mit den dazugehörigen ersetzen. So entspricht *security* in verschlüsselter Form *vhfxulwb*. Zusätzlich kann der Verschiebungsfaktor des Alphabets beliebig gewählt werden, somit auch negativ. Zudem ist es bspw. möglich ein Wort mehrmals zu verschlüsseln.

Level 5

In dem vorliegenden Passworträtsel bekommt der Einsteiger wieder eine Login-Maske (Siehe Abbildung 8.9) vorgegeben mit dem Hinweis, dass das Passwort 'LyRhtZhoBukZplnal' lautet. Allerdings ist dieses mit der Caesar-Verschlüsselung encodiert worden.



> Irgendwie ist es dir gelungen an das Passwort zu gelangen, welches 'LyRhZhoBukZpInal' laut. Allerdings ist dies verschlüsselt. Finde eine Möglichkeit um das entschlüsselte Passwort zu gelangen

☐ Remember me

[Lost Password?](#)
[Register](#)

Abbildung 8.9: Login-Maske der Caesar-Verschlüsselung im fünften Level

Um herauszufinden wie das decodierte Passwort aussieht muss der Einsteiger folglich die Verschiebung des Alphabets herausfinden. So könnte der Einsteiger z. B. ein eigenes Skript schreiben oder einen Caesar-Verschlüsselungsprogramm (<https://gc.de/gc/caesar/>) nutzen und jeden Verschiebungsfaktor durch zu spielen bis ein sinnvolles Passwort herauskommt. Bei dem Durchexerzieren sollte erkenntlich werden, dass bei einer Verschiebung um den Faktor -7, die Lösung für dieses Level herauskommt. Das korrekt entschlüsselte Passwort lautet `ErKamSahUndSiegte`

8.2.6 Level 6 - Wireshark-Sniffing

Im sechsten Level des Parcours soll der Einsteiger in Kontakt mit Wireshark kommen. Die Abbildung 8.10 zeigt die Aufgabenstellung.

➤ Mit Wireshark ist es möglich den Netzwerkverkehr zu beobachten und Pakete zu verfolgen. Die folgende PCAP Datei enthält unverschlüsselten FTP-Verkehr in die Credentials für dieses Level enthalten sind. Finde Sie und löse die Aufgabe! (PCAP Mitschnitt)

Username

Password

☐ Remember me

Login

[Lost Password?](#)
[Register](#)

Abbildung 8.10: Login-Maske des sechsten Levels

Hier soll der Einsteiger eine unverschlüsselte PCAP-Datei auslesen, die einen Nachrichtenverkehr wiedergibt. In dieser Datei ist an einer Stelle, der Username mit Passwort auslesbar. Abbildung 8.11 zeigt den relevanten Ausschnitt in Wireshark. Die Lösung des Levels ist der Username `jerry` und das Passwort `saymynameheisenberg`.

2 0.002099	192.168.1.10	192.168.1.10	TCP	74 21 → 52796 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=1400211 TSecr=144544
3 0.002137	192.168.1.10	192.168.1.10	TCP	66 52796 → 21 [ACK] Seq=1 Ack=1 Win=14608 Len=0 TSval=144544 TSecr=1400211
4 0.004071	192.168.1.10	192.168.1.10	FTP	81 Response: 220- smallftpd
5 0.004159	192.168.1.10	192.168.1.10	TCP	66 52796 → 21 [ACK] Seq=1 Ack=16 Win=14608 Len=0 TSval=144545 TSecr=1400211
6 0.005103	192.168.1.10	192.168.1.10	FTP	169 Response: 1.0.3
7 0.005151	192.168.1.10	192.168.1.10	TCP	66 52796 → 21 [ACK] Seq=1 Ack=119 Win=14608 Len=0 TSval=144545 TSecr=1400211
8 11.046516	192.168.1.10	192.168.1.10	FTP	78 Request: USER jerry
9 11.048725	192.168.1.10	192.168.1.10	FTP	100 Response: 331 User name okay, password required.
10 11.048797	192.168.1.10	192.168.1.10	TCP	66 52796 → 21 [ACK] Seq=13 Ack=159 Win=14608 Len=0 TSval=147306 TSecr=1401315
11 15.884315	192.168.1.10	192.168.1.10	FTP	92 Request: PASS saymynameheisenberg
12 15.886934	192.168.1.10	192.168.1.10	FTP	88 Response: 230 User logged in.
13 15.887005	192.168.1.10	192.168.1.10	TCP	66 52796 → 21 [ACK] Seq=39 Ack=181 Win=14608 Len=0 TSval=148516 TSecr=1401799

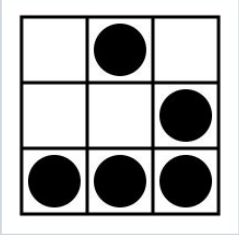
➤ Frame 8: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
 ➤ Ethernet II, Src: Foxconn_32:00:10:00:15:58 (00:15:58:32:00:10), Dst: Asustek_af:77:45 (f4:6d:04:af:77:45)
 ➤ Internet Protocol Version 4, Src: 192.168.1.106, Dst: 192.168.1.10
 ➤ Transmission Control Protocol, Src Port: 52796, Dst Port: 21, Seq: 1, Ack: 119, Len: 12
 ➤ File Transfer Protocol (FTP)

Abbildung 8.11: Wiresharkausschnitt für die Lösung des sechsten Levels

8.2.7 Level 7 - BCrypt-Verschlüsselung

Das letzte Level umfasst einen weiteren Verschlüsselungsart - der BCrypt-Verschlüsselung. BCrypt ist eine kryptologische Hashfunktion, die speziell für das Hashen und Speichern von Passwörtern entwickelt wurde. Für dieses Level wurde die eingebaute BCrypt Funktion von PHP verwendet.

In der Aufgabe findet der Einsteiger eine Login-Maske (Siehe Abbildung 8.12 wieder, in der eine Tabelle mit verschlüsselten Passwörtern enthalten ist. Zudem ist eine Liste von sogenannten *Common-Passwords* gegeben, die häufig genutzte Passwörter wieder spiegelt. Somit ist die Aufgabe die gegebenen Passwörter zu entschlüsseln und zu überprüfen ob diese mit einem, der *Common-Passwords* übereinstimmt.



> Du warst in der Lage mittels einer SQL-Injection an die Tabelle mit Usernamen und Passwörtern zu gelangen. Allerdings sind dort nur verschlüsselte Werte vorzufinden. Finde einen Weg in das System! Im ersten Schritt gehst du davon aus, dass min. ein User so freundlich gewesen ist und eines der gebräuchlichen Passwörter zu benutzen.
 ([commonpasswords.txt](#))

ID	Username	Hash
0	Daniel	\$2y\$10\$zwynnuplPCUSxUKAt2jK5.ht\$Xwvkn/el3zMb6ZdgBMdmCuWobWXO
1	Mohammed	\$2y\$10\$GXOsE7j7Qs40FdmuiDorUuprsvPaZFqms.iGCbLzCaLYMveAvxNPS
2	Stefan	\$2y\$10\$b6XHaEhR0Zwqaf6.CTYkv.QdU2flh796DxB24wjyXoYv49GuBt4pC
3	Christian	\$2y\$10\$e5yfZ.2GxpQqaw740ZvjAeZ/IMbi2pOV8IsH3DwN0uaMBVALDiO7C
4	Kevin	\$2y\$10\$roMOvhk2uJ.9QMbUtKgHZOVflv/EsXsfpoBWEuN1WAvAIJBb5D
5	Sebastian	\$2y\$10\$K7Zz/co4Fjh1V.QGU.K/Gua3IRdR8HbnYnsVwqoyAejBclziuBR5Ta
6	Fabian	\$2y\$10\$OM/uZQ5PAdHFvOTFgO8y0OMciHWKW2PPALsM40koeP9Dzdvi5BhJS

Username
 Password
☐ Remember me

Login

[Lost Password?](#)
[Register](#)

Abbildung 8.12: BCrypt Login Maske mit gegebenen Credentials

Die Lösung für dieses Level lautet: Username `Kevin` und Passwort `sniffing`.

8.3 Ausblick

In Bezug auf die künftigen Entwicklungsschritte des Login Parcours sollen nun einige Vorschläge gemacht werden.

Die bisherigen Level orientieren sich zum Teil an Beispielübungen der Seite <http://oxf.at/>. Dort ist eine ähnliche Aufgabensammlung aufgezeigt, die als Inspi-

ration gelten kann. Denkbar ist es auch, den Login Parcours in Richtung echter Hacker-Rätsel wie z. B. auf <https://www.root-me.org/> aufzubauen. Diesbezüglich kann dies auch zusammen mit der Austragung des eigenen CTFs geschehen, der zum ersten Mal im Februar 2018 stattfindet.

9 SQL-Injection

Eine SQL-Injection ist ein Angriff auf eine Benutzerschnittstelle, die mit einer Datenbank im Hintergrund kommuniziert. Dabei werden SQL-Befehle z.B. über die normalen Eingabefelder einer (Web-)Applikation an die Datenbank geschickt und dort ausgeführt. Dies kann dazu führen, dass der Angreifer Zugriff auf sensible Daten oder Anwendungen erhält oder sogar die komplette Datenbank löschen kann.

9.1 Erklärung

Beinahe jede moderne Anwendung - sei es eine Webanwendungen wie Facebook oder eine klassische Client-Server-Applikation mit einer speziellen Benutzeroberfläche wie SAP ERP - verwendet im Hintergrund ein Datenbankmanagementsystem zur Verwaltung und Speicherung der Applikationsdaten. Die Datenbank ist dabei i.d.R. von größerem Wert als die Anwendung selbst. In Industrieunternehmen enthalten Datenbanken z.B. Informationen zu Mitarbeitern, Kunden, Finanztransaktionen, Produktionsplänen oder geheime Dokumente der Produktentwicklung. Datenbanken sind somit ein kritischer Bestandteil vieler Unternehmen. Deren Verfügbarkeit und Sicherheit ist wichtig für den Fortbestand des Unternehmens und daher auch gesetzlich geregelt¹.

Kriminell motivierte Hacker haben daher ein hohes Interesse daran, Zugang zu diesen Daten zu erhalten. Eine möglicher Zugriffsweg hierfür ist das Ausnutzen von Schwachstellen durch SQL-Injections.

Um SQL-Injections durchführen zu können, wird lediglich ein grundlegendes Verständnis klassischer Anwendungsarchitekturen und der Datenbankabfragesprache SQL benötigt. Die Grundlagen hierzu werden nachfolgend erläutert.

9.1.1 Grundlagen Datenbanksysteme

Datenbanksysteme (DBS) sind ein weithin genutztes Hilfsmittel zur rechnergestützten Organisation, Erzeugung, Veränderung und Verwaltung großer Daten-

¹Siehe https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/baust/b05/b05007.html

sammlungen und stellen in vielen Unternehmen und Organisationen die zentrale Informationsbasis zu ihrer Aufgabenerfüllung bereit. Ein DBS besteht aus einem *Datenbankmanagementsystem* (DBMS) und einer oder mehrerer Datenbanken. Eine Datenbank ist eine Zusammenstellung von Daten samt ihrer Beschreibung (Metadaten), die persistent im DBS abgelegt werden.

Das DBMS bildet die Schnittstelle zwischen den Datenbanken und dient den Benutzern zur Datenverwaltung und -Veränderung. Die zentralen Aufgaben eines DBMS sind im Wesentlichen die Bereitstellung verschiedener Sichten auf die Daten (Views), die Konsistenzprüfung der Daten (Integritätssicherung), die Autorisationsprüfung, die Behandlung gleichzeitiger Zugriffe verschiedener Benutzer (Synchronisation) und das Bereitstellen einer Datensicherungsmöglichkeit, um im Falle eines Systemausfalls zeitnah Daten wiederherstellen zu können.

Der Zugriff auf die Daten erfolgt mithilfe einer standardisierten Abfragesprache, der *Structured Query Language* (SQL). Durch sie können Datenstrukturen angelegt und verändert werden, neue Daten zur Datenbank hinzugefügt sowie bestehende Daten verändert oder gelöscht werden.

9.1.2 3-Schichten-Architektur

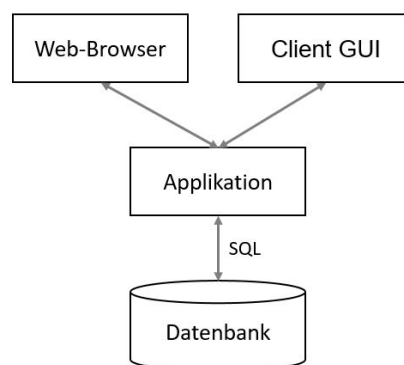


Abbildung 9.1: 3-Schichten-Architektur

DBS werden von Endanwendern nicht direkt genutzt, sondern werden durch die Applikation und graphische Oberflächen verschalt. Der Benutzer greift z.B. via HTTP über die Oberfläche auf die Applikation zu. Die Applikation selbst ist mit einem dedizierten Datenbankbenutzer mit dem DBS verbunden, die Kommunikation erfolgt über SQL. Diese Architektur wird wegen ihrer drei Ebenen - der Präsentations-, der Logik- und der Persistenzschicht - auch als 3-Schichten-Architektur bezeichnet (vergleiche Abbildung 9.1).

Die Applikationen stellen dem Benutzer Eingabefelder zur Verfügung, mittels derer die Benutzer Daten auslesen, verändern oder neu erzeugen können. Die Benutzereingaben werden zu bereits vorgefertigten SQL-Statements hinzugefügt und an das DBS gesendet. Das DBS verarbeitet das Statement und sendet eine Antwort an die Anwendung zurück.

9.1.3 Der Angriff

Bei einer SQL-Injection werden, wie der Name schon impliziert, (Teile von) SQL-Statements an die normalen Benutzereingaben angehängt, um somit die Logik und die Sicherheitsmechanismen der Applikation zu umgehen.

Der SQL-Interpreter des DBMS führt das ursprüngliche und die angehängten Statements aus. Mittels geschickter SQL-Injections können über harmlose Benutzerschnittstellen ganze Datenbanken gelöscht werden.

9.2 Vorbereitung

Für die Ausführung des Tutorials wird Kali Linux 2.0 mit eingerichteter Security Workbench benötigt. Alternativ kann das Skript auf einem anderen beliebigen Linux-System verwendet werden, in dem MySQL und Apache2 installiert sind. Zur korrekten Initialisierung der Webanwendung muss ggf. der Pfad zum Apache-Webserver in der Datei 'initializeDB.py' geändert werden. Die zu konfigurierenden Pfade im Quellcode sind entsprechend gekennzeichnet.

9.3 Ablauf

9.3.1 Aufbau des Login-Web-Services

Das Tutorial wird über die Security Workbench unter dem Hauptmenüpunkt 4 aufgerufen. Zu Beginn wird der Apache2-Webserver und das MySQL-DBMS gestartet. Anschließend wird die Datenbank initialisiert. Dabei wird folgendes Schema erstellt:

Field	Type	Null	Key	Default	Extra
userId	int(11)	NO	PRI	NULL	auto_increment
userName	varchar(255)	YES		NULL	
password	varchar(30)	YES		NULL	

Abbildung 9.2: Tabellenstruktur der Tabelle „secretUserData“

Nun stehen verschiedene Tutorials zur Verfügung. Sie alle basieren auf demselben Web-Service, einem Login für eine Website (siehe Abbildung 9.3). Der Web-Service wurde in HTML/CSS, JavaScript und PHP entwickelt. Die Benutzereingaben werden auf der HTML-Seite entgegen genommen und über einen Ajax-Aufruf an PHP übergeben.

Login

Benutzername:

Passwort:

Abbildung 9.3: Login-Oberfläche des Web-Services

Dort werden die Benutzereingaben in ein vordefiniertes SQL-Statement eingefügt (siehe Listing 9.1) und an das DBS zur Ausführung übermittelt. Die Antwort des Servers, ein oder mehrere zutreffende Tupel mit der User-ID, dem User-Namen und dem User-Passwort werden anschließend unterhalb des Eingabefelds in dem Web-Service angezeigt. Dort ist ebenfalls das im DBS ausgeführte SQL-Statement zu sehen.

Listing 9.1: SQL-Statement

```
$query = '
    SELECT *
    FROM secretUserData
    WHERE userName = "'. $username. '"
    AND password = "'. $password. '";
';
```


Zudem sind zwei Buttons verfügbar, mit denen die Tabellenstruktur sowie der momentane Inhalt der Tabelle angezeigt werden können.

9.3.2 SQL-Injection zum Auslesen von Daten

Im ersten Teil des Tutorials werden mittels einer einfachen SQL-Injection Daten aus der Datenbank gelesen, auf die man über die Anwendung eigentlich keinen Zugriff hätte. Über die zwei Eingabefelder „Benutzername“ und „Login“ kann sich der Benutzer bei einer Anwendung anmelden. Die Eingaben werden an die Datenbank geschickt und in einem SELECT-Statement überprüft. Anschließend wird der selektierte Datensatz zurück geschickt.

Als erstes melden wir uns mit einem schon bekannten User und Passwort an, um die Funktionsweise zu testen. Nutze hierzu den User `Douglas Adams` mit dem Passwort `DontPanic!` und drücke auf den "LoginButton."

Login

Benutzername:

Passwort:

Folgende Query wurde gebildet:
 SELECT * FROM secretUserData WHERE userName = "Douglas Adams" AND password = "DontPanic!";

Login successful with user:

ID	Name	Password
1	Douglas Adams	DontPanic!

Abbildung 9.4: Normaler Login

Dieses Szenario spiegelt die angedachte Nutzung des Login-Dienstes wieder. Ein Nutzer meldet sich mit seinen Anmeldedaten an und deren Existenz wird in der Datenbank überprüft. Stimmen die Anmeldedaten überein, ist der Nutzer angemeldet und hat Zugriff auf die Anwendung.

Als nächstes sollen mittels einer SQL-Injection alle User der Datenbank „secretUserData“ ausgegeben werden. Ersetze die aktuellen Eingaben hierzu z.B. durch `blabla" OR "1"="1`.

Login

Benutzername:

Passwort:

Folgende Query wurde gebildet:
 SELECT * FROM secretUserData WHERE userName = "blabla" OR "1"="1" AND password = "blabla" OR "1"="1";

Login successful with user:

ID	Name	Password
1	Douglas Adams	DontPanic!
2	Harry Potter	CaputDraconis
3	James T. Kirk	BeamMeUpScotty
4	Grumpy Cat	No!
5	Dalek	Exterminate!
6	The Doctor	Allons-y
7	Deadpool	Chimichanga

Abbildung 9.5: Login mit SELECT-Injection

Nun wurden alle Tupel, die in der Tabelle „secretUserData“ enthalten sind ausgegeben. Möglich ist das durch das Anhängen von z.B. `OR "1"="1"` an eine beliebige Eingabe. Hierdurch werden die Abfragen der WHERE-Klausel grundsätzlich zu TRUE ausgewertet. Am obigen Beispiel erläutert bedeutet dies:

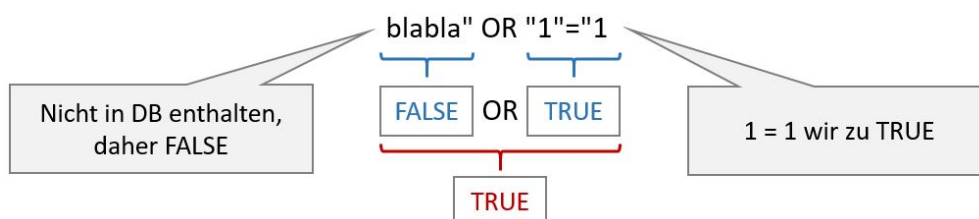


Abbildung 9.6: Auswertung von OR "1"="1"

Sobald alle Ausdrücke innerhalb der WHERE-Klausel zu TRUE evaluiert wurden, wird die gesamte Datenbanktabelle ausgegeben. Hängt man den Zusatz lediglich

an das Eingabefeld für das Passwort an, erhält man den Datensatz für den eingegebenen Benutzer. Dieses Szenario ist z.B. typisch, wenn man bereits einen möglichen Benutzernamen für die Applikation kennt, aber dessen Passwort unbekannt ist.

9.3.3 SQL-Injection zum Einfügen von Daten

Im zweiten Teil des Tutorials wird mittels einer SQL-Injection ein zusätzlicher Datensatz in die Tabelle eingefügt. Die Beispiel-Applikation ist äquivalent zu der aus dem ersten Tutorial. Dieses mal hängen wir an einen beliebigen Benutzernamen folgendes INSERT-Statement inkl. Kommentar an: `"; INSERT INTO secretUserData VALUES(1234, "Hackerman", "fsociety"); --`. Im Eingabefeld für das Passwort können ebenfalls beliebige Zeichen eingegeben werden. Durch diese Injection werden dem DBS prinzipiell drei Befehle übergeben:

Login

Benutzername:

Passwort:

Folgende Query wurde gebildet:
 SELECT * FROM secretUserData WHERE userName = "bla"; INSERT INTO secretUserData VALUES(1234, "Hackerman", "fsociety"); -- " AND password = "blabla";

Kein User mit Namen bla"; INSERT INTO secretUserData VALUES(1234, "Hackerman", "fsociety"); -- und Passwort blabla vorhanden

Hier kannst du dir die Tabellenstruktur bzw. den aktuellen Tabelleninhalt der Tabelle "secretUserData" jederzeit ansehen, um die Auswirkung deiner SQL-Injection zu prüfen.

ID	Name	Password
1	Douglas Adams	DontPanic!
2	Harry Potter	CaputDraconis
3	James T. Kirk	BeamMeUpScotty
4	Grumpy Cat	No!
5	Dalek	Exterminate!
6	The Doctor	Allons-y
7	Deadpool	Chimichanga
1234	Hackerman	fsociety

Abbildung 9.7: TODO: aktuelles Bild Login mit INSERT-Injection

- Das ursprüngliche SELECT-Statement bis zur Eingabe eines Benutzers:
`SELECT * FROM secretUserData WHERE username = "<übergebener Benutzername>";`
- Das angehängte INSERT-Statement:
`INSERT INTO secretUserData VALUES(1234, "Hackerman", "fsociety");`

- Ein Kommentar, der in SQL mit zwei Bindestrichen eingeleitet wird: `--`. Hierdurch wird der SQL-Code, der noch zum ursprünglichen SELECT-Statement gehört, als Kommentar vom SQL-Interpreter ignoriert. Im Beispiel betrifft das: `" AND password = "<übergebenes Passwort>"`;

Sieht man sich nach der Ausführung des Statements die Inhalte der Tabelle an, ist zu sehen, dass sich ein neuer Datensatz mit der User-ID 1234, dem Usernamen „Hackerman“ und dem Passwort „fsociety“ enthält. Nutzt man für die Injection zusätzlich einen existierenden Benutzernamen statt der Eingabe „bla“, wird man gleichzeitig bei der Applikation angemeldet.

9.3.4 SQL-Injection zum Löschen von Tabellen

Im dritten Teil des Tutorials wird mittels einer SQL-Injection die komplette Tabelle gelöscht (DROP).

Bitte beachte, dass du die Datenbank erst im Hauptmenü des Konsolen-Skripts im Unterpunkt „5. Datenbank zurück setzen“ wieder initialisieren musst, wenn du nach dem DROP weiterarbeiten möchtest!

Nun hängen wir an einen beliebigen Benutzernamen folgendes Statement inkl. Kommentar an: `"; DROP TABLE secretUserData; --`. Im Eingabefeld für das Passwort können ebenfalls beliebige Zeichen eingegeben werden.

Login

Benutzername:

Passwort:

Folgende Query wurde gebildet:
SELECT * FROM secretUserData WHERE userName = "Douglas Adams"; DROP TABLE secretUserData; --" AND password = "blabla";

.....

Login successful with user:

ID	Name	Password
1	Douglas Adams	DontPanic!

.....

Hier kannst du dir die Tabellenstruktur bzw. den aktuellen Tabelleninhalt der Tabelle "secretUserData" jederzeit ansehen, um die Auswirkung deiner SQL-Injection zu prüfen.

Datenbank existiert nicht

Abbildung 9.8: TODO: Bild aktualisieren Login mit DROP-Injection

Die Ausführung der drei Statements (SELECT, DROP, Kommentar) ist äquivalent zum vorherigen Beispiel.

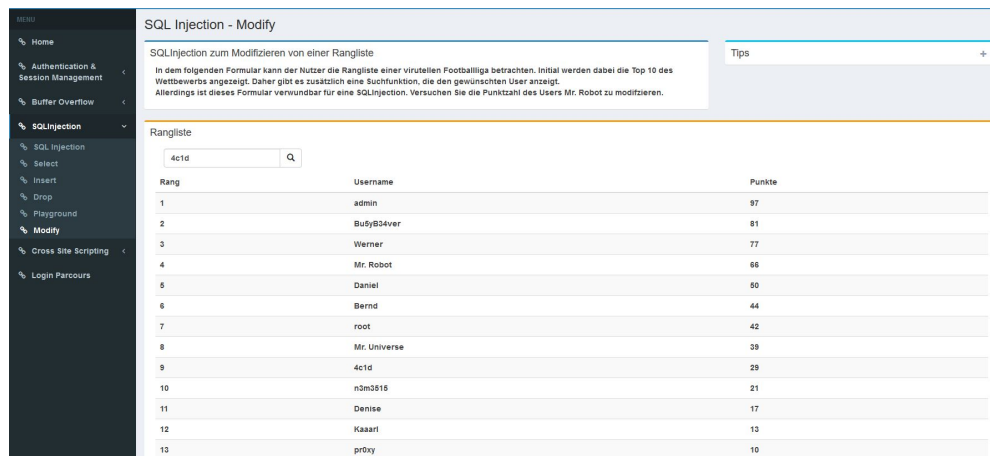
9.3.5 SQL-Injection zum Modifizieren von Tabellen

Das vierte Tutorial befasst sich mit modifizierenden SQL-Queries, die den Inhalt einer bestehenden Datenbanktabelle verändern. Technisch wird dies mit dem Update-Statement realisiert (siehe Listing 9.2).

Listing 9.2: SQL-UPDATE-Statement

```
$query = '
    UPDATE table_name
    SET column1 = value1, column2 = value2, ...
    WHERE condition;
';
```

In Bezug auf das Tutorial umfasst die Datenbank eine Tabelle `sqlRanking`, die eine virtuelle Rangliste entspricht (siehe Abbildung 9.9). Diese besitzt die Spalten Username und Punkte. Zudem wird während der Initialisierung der Webseite die Rangliste auf Basis der Punktzahl absteigend sortiert.



SQL Injection - Modify

SQLInjection zum Modifizieren von einer Rangliste

In dem folgenden Formular kann der Nutzer die Rangliste einer virtuellen Fußballliga betrachten. Initial werden dabei die Top 10 des Wettbewerbs angezeigt. Daher gibt es zusätzlich eine Suchfunktion, die den gewünschten User anzeigt. Allerdings ist dieses Formular verwundbar für eine SQLInjection. Versuchen Sie die Punktzahl des Users Mr. Robot zu modifizieren.

Tips

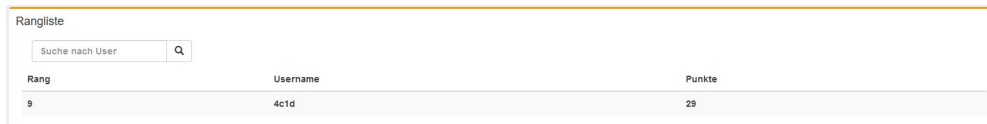
Rangliste

4c1d

Rang	Username	Punkte
1	admin	97
2	Bu9yB04ver	81
3	Werner	77
4	Mr. Robot	66
5	Daniel	50
6	Bernd	44
7	root	42
8	Mr. Universe	39
9	4c1d	29
10	n3m5815	21
11	Denise	17
12	Kaaari	13
13	proxy	10

Abbildung 9.9: Rangliste mit Suchfunktion

Als Anwendungsszenario ist es hierbei möglich, die Rangliste mit einer Suchfunktion nach bestimmten Usernamen zu filtern. Abbildung 9.10 zeigt wie der Anwender mit Hilfe des Input-Feldes nach einem gewünschten User sucht. Die Kommunikation mit der Datenbank basiert hierbei auf einem einfachen SELECT-Query.



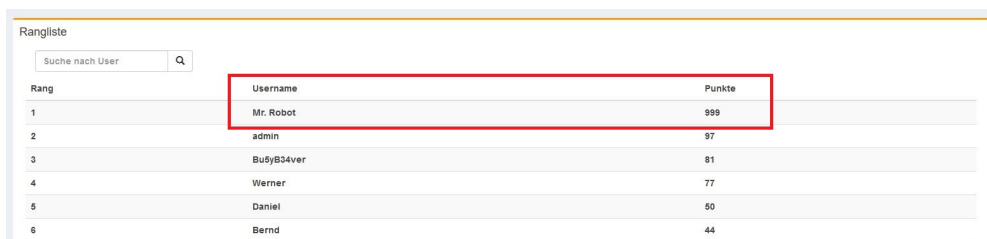
Rang	Username	Punkte
9	4c1d	29

Abbildung 9.10: Rangliste mit gesuchtem User

An dieser Stelle gilt es hervorzuheben, dass die SQL-Abfrage nicht absichert ist und somit eine Sicherheitslücke entsteht. Im Folgenden ist das Ziel des Tutorials die Punkteanzahl eines Users zu modifizieren.

Um die Aufgabe zu lösen, muss der Anwender in das Input-Feld ein SQL-Statement einfügen, das bspw.

`' ; UPDATE sqlInjectionRanking SET punkte = 999 WHERE username = 'Mr. Robot'` entspricht. Wichtig ist das die erste SELECT-Abfrage gültig ist. Abbildung 9.11 zeigt wie die Lösung des Tutorials aussehen kann.



Rang	Username	Punkte
1	Mr. Robot	999
2	admin	97
3	Bu9yB34ver	81
4	Werner	77
5	Daniel	50
6	Bernd	44

Abbildung 9.11: Rangliste mit gesuchtem User

Um das Tutorial zu vereinfachen, bekommt der Anwender je nach Wissensstand Tipps an die Hand, die ihn zur Lösung der Aufgabe hinführen sollen.

9.3.6 Die SQL-Injection-Spielwiese

Im letzten Tutorial der SQL-Injection in der Broken-Web-Anwendung findest du die SQL-Injection-Spielwiese. Innerhalb dieser Spielwiese kannst du beliebige SQL-Injections ausprobieren.

Szenario	Statement	Einfügen in...
SELECT	bla" OR "1"="1	Benutzername und/oder Passwort
INSERT	bla"; INSERT INTO secretUserData VALUES(1234, "Hackerman", "fsociety"); --	Benutzername oder Passwort
UPDATE	bla"; UPDATE secretUserData SET password = "0000"; --	Benutzername oder Passwort
ALTER TABLE	bla"; ALTER TABLE secretUserData ADD COLUMN blabla INT; --	Benutzername oder Passwort
DROP TABLE	bla"; DROP TABLE secretUserData; --	Benutzername oder Passwort
DROP SCHEMA	bla"; DROP SCHEMA vulnerableDB; --	Benutzername oder Passwort
CREATE TABLE	bla"; CREATE TABLE NewTable(Col1 INT PRIMARY KEY, Col2 INT); --	Benutzername oder Passwort

Abbildung 9.12: TODO: aktuelles bild Verschiedene SQL-Injections zum Ausprobieren

Als Hinweis sind die Statements der vorhergegangenen Beispiele und einige Weitere im nachfolgenden Fenster hinterlegt. Um sie zu sehen musst du lediglich den Inhalt des Fensters mit der Maus markieren. Bitte beachte, dass du die Datenbank im Hauptmenü des Konsolen-Skripts im Unterpunkt „5. Datenbank zurück setzen“ wieder initialisieren musst, wenn du nach einer Datenstruktur verändernden SQL-Injection weiter arbeiten willst. Dazu musst du die Anwendung nicht schließen. Deine Änderungen am Inhalt oder an der Struktur der Datenbank kannst du mit der Anzeige der Tabellenstruktur bzw. dem Inhalt jederzeit prüfen.

Neben den hier aufgelisteten gibt es noch eine Vielzahl weiterer SQL-Injections. Nicht alle werden in diesem Beispiel funktionieren, da der Erfolg von SQL-Injections von mehreren Faktoren abhängig ist:

- Die Programmiersprache der Anwendung
- Das verwendete DBMS
- Die Berechtigungen, die der Datenbankbenutzer der Anwendung inne hat

9.4 Gegenmaßnahmen

Viele Programmiersprachen haben mittlerweile Mechanismen eingebaut, mittels derer SQL-Injections abgewehrt werden können. So ist es in den meisten Programmier- und Skriptsprachen nicht mehr möglich, innerhalb eines DB-Aufrufs mehrere SQL-Statements auszuführen. In einigen wenigen ist dies immer noch möglich, wie z.B. in der Skriptsprache PHP.

9.4.1 Prepared Statements

Durch sogenannte „Prepared Statements“ können SQL-Injections vollständig unterbunden werden. Statt das SQL-Statement komplett auf der Applikationsseite

zusammenzustellen, wird das Statement auf zwei Mal an das DBS gesendet. Im ersten Aufruf wird das vorbereitete Statement ohne die Nutzereingaben an das DBS übermittelt. Hierdurch wird dem DBS die Struktur des Statements im Vorfeld angekündigt. Nachfolgend ist das Prepared Statement eines SELECT-Statements zu sehen:

Listing 9.3: Prepared Statement

```
prepare("SELECT * FROM secretUserData where userName = ?");
```

Die vom Nutzer eingegebenen Parameter werden erst im Anschluss an das DBS übermittelt:

Listing 9.4: Übergabe der Parameter

```
execute($_GET['name']);
```

Dort werden sie in das bereits angekündigte Statement eingefügt und ausgeführt. Übergebene Parameter, auch wenn ihnen SQL-Code hinzugefügt wurde, werden ausschließlich als Textinput interpretiert.

9.4.2 Escapen von Eingaben

Eine weitere Möglichkeit die Datenbank vor unberechtigtem Zugriff und Manipulationen zu schützen ist das Escapen aller Nutzereingaben. Das grundlegende Problem bei SQL-Injections ist die Interpretation von Texteingaben als ausführbare Anweisungen für das DBMS. Durch das Escapen der Eingaben werden Metazeichen wie Anführungszeichen maskiert und somit vom SQL-Interpreter nicht beachtet. Nachfolgend ist das Escapen von Strings in PHP dargestellt:

Listing 9.5: Escapen von Strings in PHP

```
mysql_real_escape_string("some String");
```


10 Disclaimer

Das vorliegende Dokument und die Broken Web Applikation sind im Rahmen eines Projektes des Masterstudiengangs Informatik an der Technischen Hochschule Ingolstadt (THI) im Wintersemester 2017/18 entstanden....