# SQL Injection

## Omówienie i prewencja w Django

FUDO SECURITY

# Adam Wołk


a.wolk@fudosecurity.com

awolk@openbsd.org

https://blog.tintagel.pl

@mulander


# SQL Injection

?

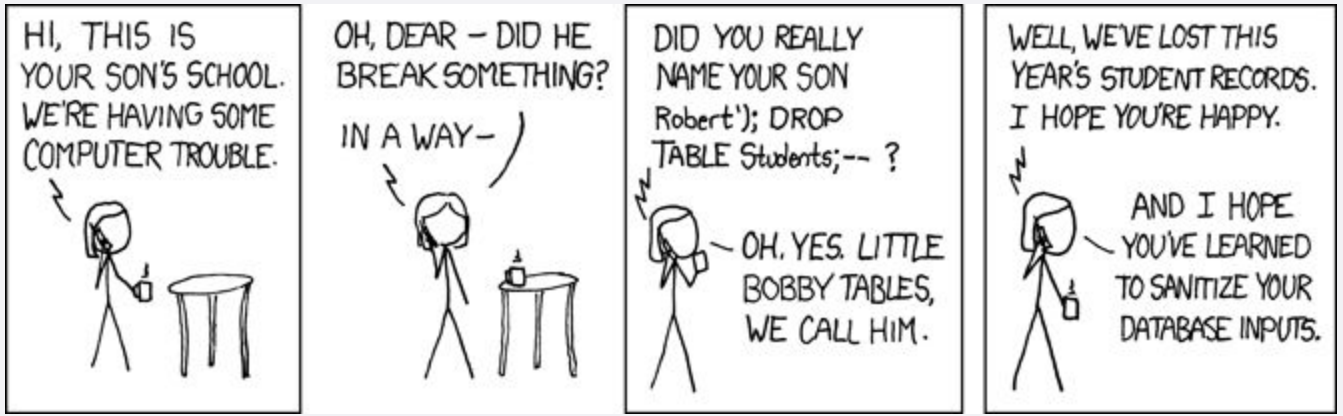
# SQL Injection

Nadużycie zaufania:

- Bazy danych wobec aplikacji
- Aplikacji wobec użytkownika

# OWASP TOP 10 2017

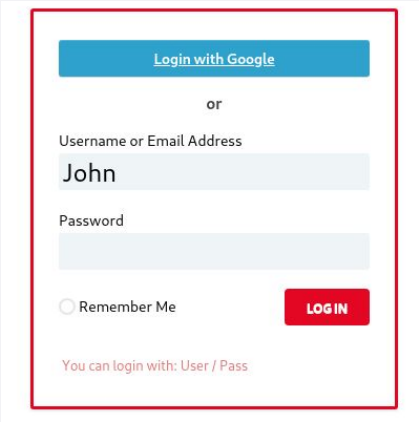
https://www.owasp.org/index.php/Top_10-2017_Top_10
https://www.owasp.org/index.php/Top_10-2017_A1-Injection




https://xkcd.com/327/

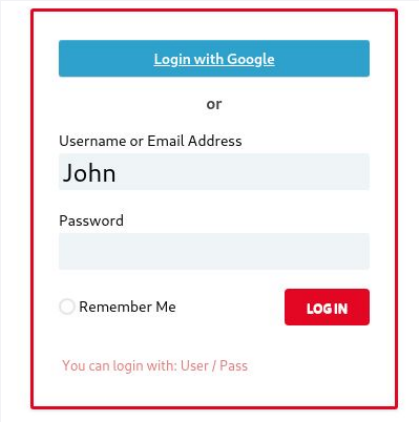
**Robert’); DROP TABLE Students; --**

```
cur.execute(
    “INSERT INTO Students(name) VALUES(‘%s’)” % name
)
```




```
INSERT INTO Students(name)
VALUES(‘John’);
```

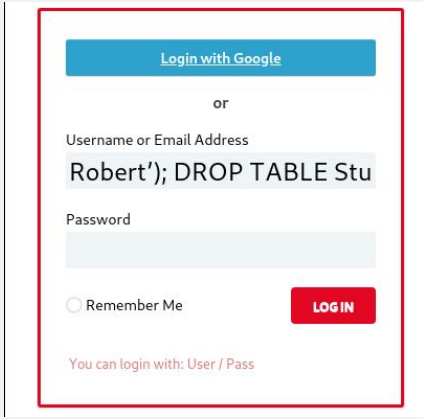
```
cur.execute(
    “INSERT INTO Students(name) VALUES(‘%s’)” % name
)
```




```
INSERT INTO Students(name)
VALUES(‘Robert’); DROP TABLE Students; --);
```

```
INSERT INTO Students(name) VALUES(‘Robert’);
```

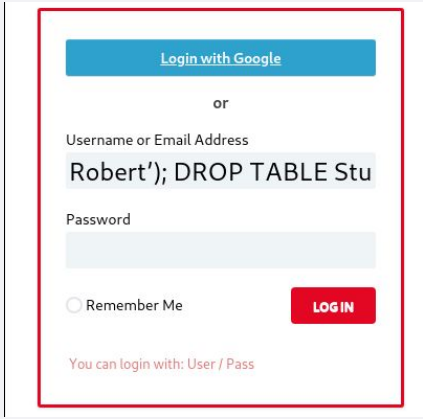
```
cur.execute(
    “INSERT INTO Students(name) VALUES(‘%s’)” % name
)
```




```
INSERT INTO Students(name)
VALUES(‘Robert’); DROP TABLE Students; --);
```

```
INSERT INTO Students(name) VALUES(‘Robert’);
DROP TABLE Students;
```

```
cur.execute(
    “INSERT INTO Students(name) VALUES(‘%s’)” % name
)
```




```
INSERT INTO Students(name)
VALUES(‘Robert’); DROP TABLE Students; --);
```

```
INSERT INTO Students(name) VALUES(‘Robert’);
DROP TABLE Students;
--);
```

http://example.com/app/accountView?id=' or '1'='1

**http://example.com/app/accountView?id=' or '1'='1**

```
cur.execute("SELECT * FROM account_view WHERE id = '%s'" % id)
```

**http://example.com/app/accountView?id=' or '1'='1**

```
cur.execute(“SELECT * FROM account_view WHERE id = ‘%s’” % id)

SELECT * FROM account_view WHERE id = ‘’ or ‘1’=’1’
```

# Narzędzia atakującego

https://github.com/sqlmapproject/sqlmap/wiki/Techniques

- Boolean based blind
- Time-based blind
- Error-based
- UNION-query based
- **Stacked queries**
- Out of band
- I wiele innych...

# '; Demo();--

```
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: id=1 AND (SELECT 2980 FROM(SELECT COUNT(*),CONCAT(0x716a706271,(SELECT (ELT(2
980=2980,1))),0x7176786a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP
 BY x)a)

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SLEEP)
    Payload: id=1 AND (SELECT * FROM (SELECT(SLEEP(5)))MVIi)

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x716a706271,0x644247784b624c4b55514e646867
58706f704c634d776c5461536f526663596a6166757a4451754b,0x7176786a71),NULL-- GseO
---
[17:22:30] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5.0
[17:22:30] [INFO] fetching tables for database: 'testdb'
Database: testdb
[1 table]
+-------+
| users |
+-------+

[17:22:30] [INFO] fetched data logged to text files under '/home/stamparm/.sqlmap/output/d
ebiandev'
stamparm@beast:~/Dropbox/Work/sqlmap$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/g
et_int.php?id=1" --batch --dump -T users -D testdb
```

Recorded with asciinema

# Nie tylko bezpośrednio wprowadzane dane

- LISTEN/NOTIFY
- Upload
- Corrupt data re-use
- OCR
- I wiele więcej...


ZU 0666', 0, 0); DROP DATABASE TABLICE;--

# Prewencja

# Zalecenia ogólne

- Dobre **API** - bind params
- **white-list** - walidacja
- Unikanie **dynamicznego** SQL
- **LIMIT** - prewencja/utrudnienie wycieków
- **Przeglądy kodu**
- **Analiza statyczna** kodu

# Analiza statyczna

Bandit (https://github.com/PyCQA/bandit)

```
bandit -r -t B608,B611,B610 -f html -n 5 -o ~/report.html <path_to_sources>
```

**hardcoded_sql_expressions:** Possible SQL injection vector through string-based query construction.
**Test ID:** B608
**Severity:** MEDIUM
**Confidence:** LOW
**File:** bsides/app/models.py
**More info:** https://bandit.readthedocs.io/en/latest/plugins/b608_hardcoded_sql_expressions.html

```
40                  FROM "{}"
41                  WHERE id=%s
42              '''.format(model_name)
43              with connection.cursor() as cursor:
44                  cursor.execute(query, [self.object_id])
```

# Zalecenia ogólne

- Preparowane zapytania
- Procedury składowane
- Widoki
- Ograniczenie uprawnień
- SET search_path (PostgreSQL)

# Django

# Django - ORM

https://docs.djangoproject.com/en/2.1/topics/security/#sql-injection-protection

Tam gdzie może pojawić się surowy SQL

- RawSQL - `raw()`
- `extra()`
- Custom expressions
  - `**extra, **extra_context`
  - `as_sql()`
- Custom SQL

# Django - RawSQL

https://docs.djangoproject.com/en/2.1/topics/db/sql/#passing-parameters-into-raw

## Bad

```
User.objects.raw("select * from users where username = '%s'", (' or 1=1--',))
User.objects.raw("select * from users where username = '%s'" % "' or 1=1--")
```

## Good

```
User.objects.raw("select * from users where username = %s", param_tuple)
User.objects.raw("select * from users where username = %(name)s", param_dict)
```

# Django - extra()

https://docs.djangoproject.com/en/2.1/ref/models/querysets/#django.db.models.query.QuerySet.extra

## Bad - ‘%s’

```
User.objects.extra(select={val': "select 1 from security where username = ‘%s’"},
                   select_params=params_tuple)
```

## Good - %s

```
.extra(select=, select_params) == .raw(“SQL”, select_params)
.extra(where=, params) == .raw(“SQL”, params)
```

# Django - extra()

## Bad

```
User.objects.extra(tables=['dummy"; select \'inject\' as username, true as admin, 3 as id from users;--']).all())
<QuerySet [<User: User object (inject)>, <User: User object (inject)>]>
```

## Good

```
User.objects.extra(tables=[psycopg2.sql.Identifier(x).as_string(conn) for x in args])
```

# Django - **extra, **extra_context, as_sql()

https://docs.djangoproject.com/en/2.1/ref/models/expressions/#avoiding-sql-injection-in-query-expressions

- template - zawsze bez ‘
- `Func.__init__(self, *expressions, output_field=None, **extra)`

# Django - **extra, **extra_context, as_sql()

https://docs.djangoproject.com/en/2.1/ref/models/expressions/#django.db.models.Func.as_sql

- template - zawsze bez ‘
- `Func.as_sql(self, compiler, connection, function=None, template=None, arg_joiner=None, **extra_context)`

# Django - custom SQL

https://docs.djangoproject.com/en/2.1/topics/db/sql/#executing-custom-sql

```
>>> from django.db import connection
>>> with connection.cursor() as cursor:
...       cursor.execute("select * from users")
...       cursor.fetchall()
...
[('admin', True, 1), ('user', False, 2)]
```

# Custom SQL == psycopg2

http://initd.org/psycopg/docs/usage.html#sql-injection

## Good

```
c.execute("select count(1) from users where user = %s", ('admin',))
c.execute("select count(1) from users where user = %(name)s", {‘name’: 'admin'})
```

# custom SQL == psycopg2

http://initd.org/psycopg/docs/usage.html#sql-injection

## Bad

```
c.execute("select count(1) from users where user = ‘%s’", ('admin',))
c.execute("select count(1) from users where user = ‘%(name)s’", {‘name’: 'admin'})
c.execute("select count(1) from users where user = %s" % ('admin', ))
c.execute("select count(1) from users where user = " + 'admin')
c.execute(f"select count(1) from users where user = {admin}")
c.execute("select count(1) from users where user = {}".format(admin))
```

# Bad API design

Parametry są **opcjonalne**!

```
cursor.execute(SQL, params)
```

**Jeden znak** dzieli poprawne użycie od **SQL injection**

```
execute("" , (tuple,))
execute("" % (tuple,))
```

# Dynamiczne zapytania

# Dynamiczny SQL

http://initd.org/psycopg/docs/sql.html

```
from psycopg2 import sql
cur.execute(
     sql.SQL("INSERT INTO {} VALUES (%s, %s) LIMIT {}").format(
         sql.Identifier('users'), sql.Literal(13)
     ),
     [10,20])
>>> str.format != sql.SQL.format
True
```

# Dynamiczny SQL

http://initd.org/psycopg/docs/sql.html

```
>>> sql.SQL.__add__
<function Composable.__add__ at 0x803332840>
>>> str.__add__ != sql.SQL.__add__
True
>>> (sql.SQL("SELECT ") + sql.Literal(1)).as_string(connection.connection)
'SELECT  1'
```

# SQLAlchemy

# SQLAlchemy

Tam gdzie może pojawić się surowy SQL

- filter
- having
- distinct
- group_by
- order_by
- inne...

## Good defaults

```
session.query(User).order_by('id; select 1 as id;').all()

sqlalchemy.exc.CompileError: Can't resolve label reference for
ORDER BY / GROUP BY. Textual SQL expression 'id; select 1 as
id;' should be explicitly declared as text('id; select 1 as
id;')
```

## Footgun in the hint

```
session.query(User).order_by('id; select 1 as id;').all()

sqlalchemy.exc.CompileError: Can't resolve label reference for
ORDER BY / GROUP BY. Textual SQL expression 'id; select 1 as
id;' should be explicitly declared as text('id; select 1 as
id;')
```

## Don’t do this

```
session.query(User).order_by(text('id; select 1 as id;')).all()
```

```
(sqlalchemy) $ python test.py
[<__main__.User object at 0x802b19b00>]
```

# Podsumowanie

# “lesson=‘%s’” % “DON’T DO THIS”

Co należy zapamiętać?

- Nie używamy `“” % param`
- Nie używamy `“{}”.format(param)` oraz `f“{param}”`
- Nie używamy konkatenacji operatorem `+`
- Nie otaczamy `%s` apostrofem - “`tu jest ‘%s’ błąd`”
- Nie ufamy danym od użytkownika / innych aplikacji
- Dynamicznego SQL unikamy, jeżeli nie ma opcji to używamy **`psycopg2.sql`**

# Dziękuje!


## Adam Wołk

a.wolk@fudosecurity.com

awolk@openbsd.org

https://blog.tintagel.pl

@mulander