ASPIRA®
UNIVERSITY COLLEGE

KOLEGIJ

Sigurnost informacijskih sustava

Laboratorijska vježba br.7 – SQL Injection

Teme:

- Kali Linux
- DVWA
- SQLmap

U ovoj laboratorijskoj vježbi ćemo raditi na iskorištavanju ranjinovsti koja je prema OWASP istraživanju broj 1 u odnosu na sve sigurnosne rizike kod aplikacija. Riječ je o *SQL Injection* napadu.

SQL injection

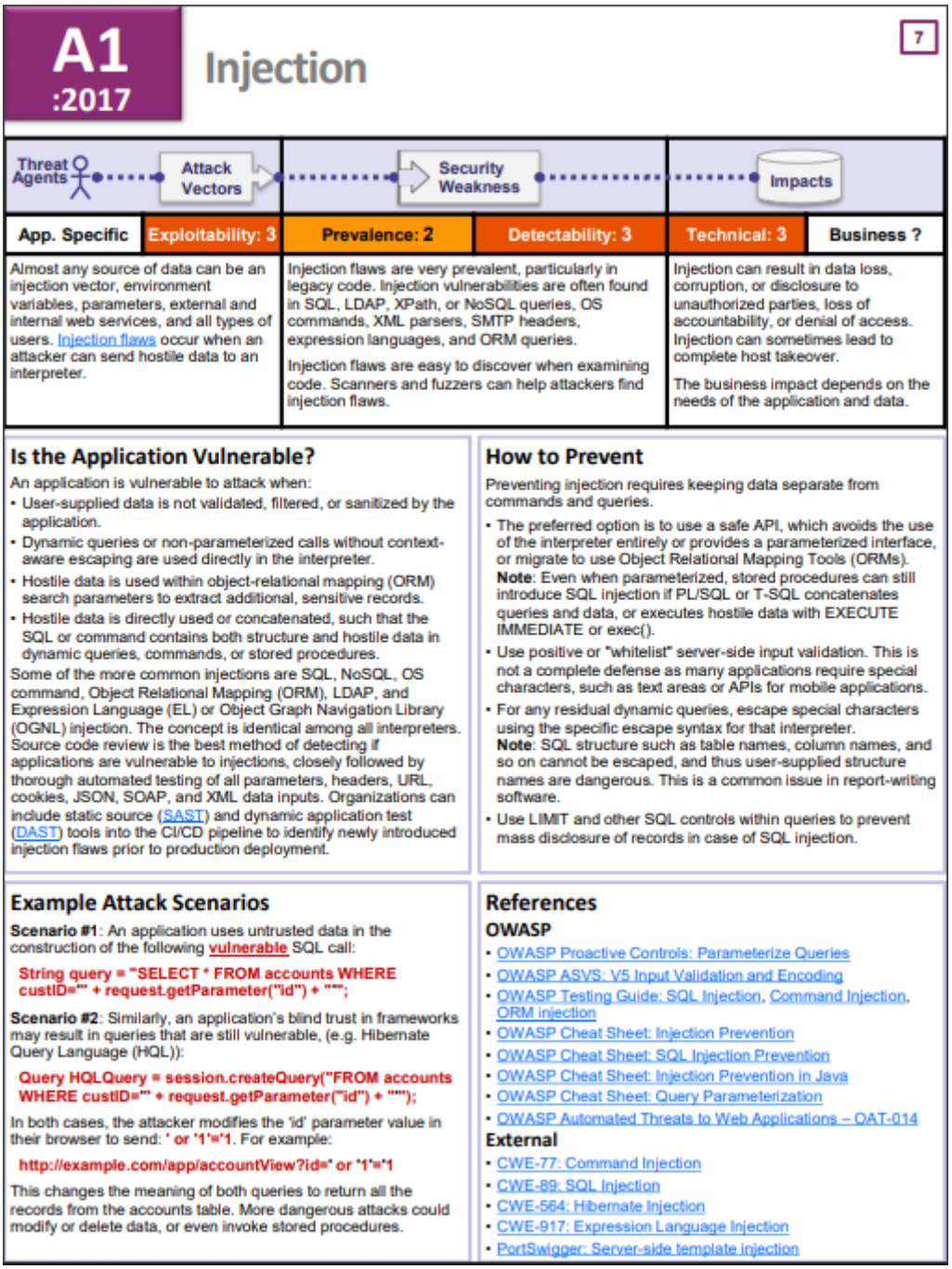
SQL *injection* ili SQL ubrizgavanje je tehnika koja omogućuje protivniku da umetne proizvoljne SQL naredbe u upite koje web aplikacija postavlja u svoju bazu podataka. Može raditi na ranjivim web-stranicama i aplikacijama koje koriste sigurnosnu bazu podataka poput MySQL, PostgreSQL, Oracle ili MSSQL.

Uspješan napad može dovesti do neovlaštenog pristupa osjetljivim informacijama u bazi podataka ili do promjene unosa (dodavanje / brisanje / ažuriranje), ovisno o vrsti baze podataka. Također, postoji mogućnost korištenja *SQL Injection* napada u svrhu zaobilaska autentifikacije i autorizacije u aplikaciji, gašenja ili pak brisanja cijele baze podataka.

Proći ćemo konkretne primjere u više tehnika koje se mogu koristiti iza iskorištavanje SQL *Injection* ranjivosti.

Cilj napada kao i u prošloj vježbi će biti aplikacija Damn Vulnerable Web Application (DVWA) koja sadržava više tipova ranjivosti između ostalog i SQL Injection ranjivosti.

Niže se nalazi izlist iz OWASP istraživanja po pitanju SQL Injection napada, tj. ranjivosti.



A1 :2017 Injection

7

Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

Is the Application Vulnerable?

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source (SAST) and dynamic application test (DAST) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
  Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive or "whitelist" server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
  Note: SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'='1. For example:

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

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

References

OWASP

- OWASP Proactive Controls: Parameterize Queries
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, ORM Injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

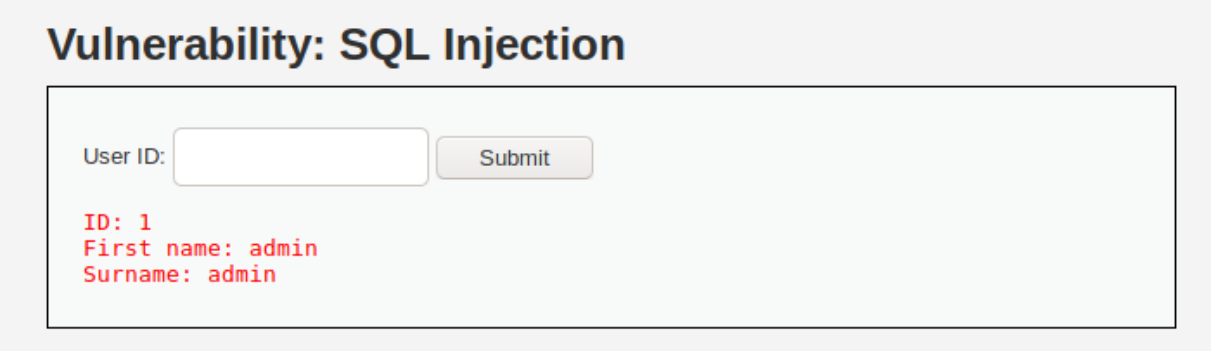
External

- CWE-77: Command Injection
- CWE-89: SQL Injection
- CWE-564: Hibernate Injection
- CWE-917: Expression Language Injection
- PortSwigger: Server-side template injection

SQL injection napad

Jedna od najčešćih vrsta SQL Injection ranjivosti koju je vrlo lako odrediti. Oslanja se na unošenje neočekivanih naredbi ili nevaljanog unosa, obično putem korisničkog sučelja, zbog čega poslužitelj baze podataka odgovara s pogreškom koja može sadržavati detalje o cilju: strukturi, verziji, operativnom sustavu, pa čak i vratiti potpune rezultate upita.

U primjeru niže, web stranica omogućuje dohvaćanje imena i prezimena korisnika za dani ID. Unošenjem 1 kao unosa za User ID, aplikacija vraća korisničke podatke iz baze podataka.



Query koji se koristi za dohvat korisnika iz baze možete pronaći u low.php file-u koji se nalazi u /var/www/html/DVWA/vulnerabilities/sqli/source direktoriju.

Query zapišite ovdje i izvrtite ga direktno na bazi na način da potražite podatke s ID-om 5 i priložite screenshot.

```
root@kali:~# mysql -u root -p dvwa
```

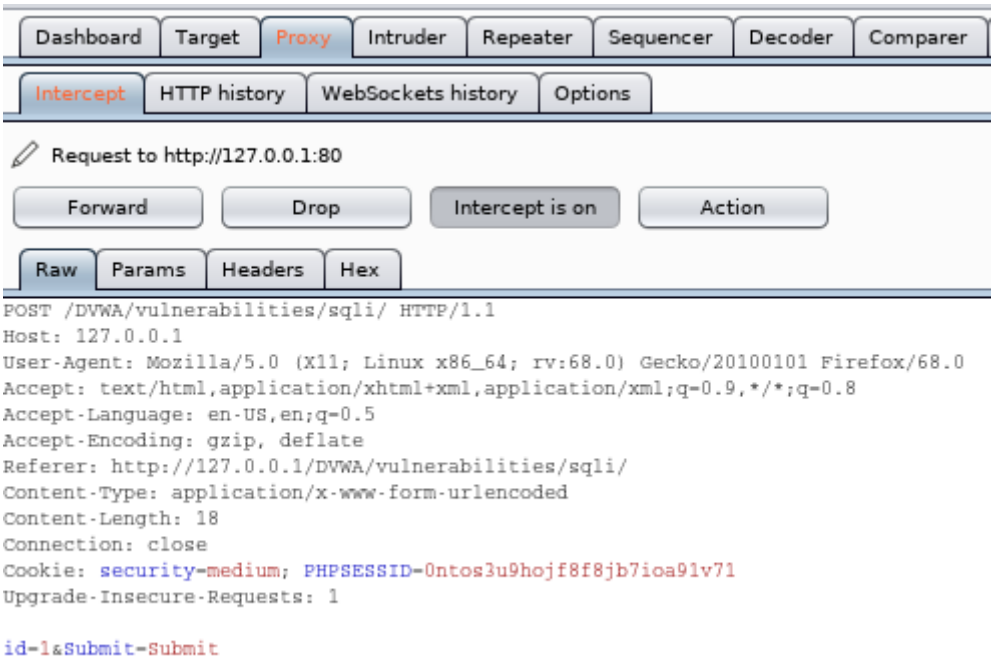
Unesite u web formu sljedeće: **1' OR '1' = '1.**

Objasnite što se dobilo s navedenim query-em, pogledajte članak o ovom tipu iskorištavanja i detaljno navedite što se desilo kao i query koji se izvršio na bazi kad se ubacio gore navedeni input. Objasnite što točno što se izvršilo da bi se dobio rezultat svih zapisa u users tablici na bazi podataka.

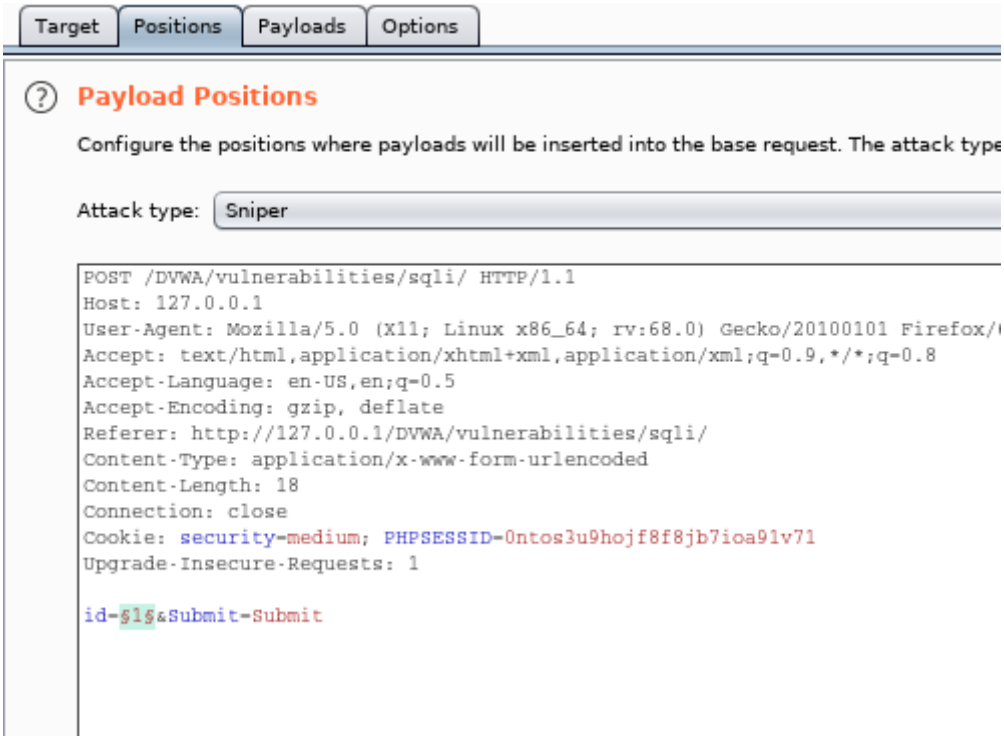
ZADATAK 2:

NVWA postavite na *medium* security. Objasnite što se promijenilo i zašto je povećana razina sigurnosti u odnosu na low security. Sve radite s ID 5.

Za ovaj zadatak koristimo BurpSuite kao i u prošloj vježbi s malom prilagodbom. Uhvatite request preko proxy načina rada koji se šalje preko web forme.



Kliknite „Sent to intruder". U intruder tabu, positions podtab odaberite attack type „sniper", odaberite ID 5 i kliknite ADD.



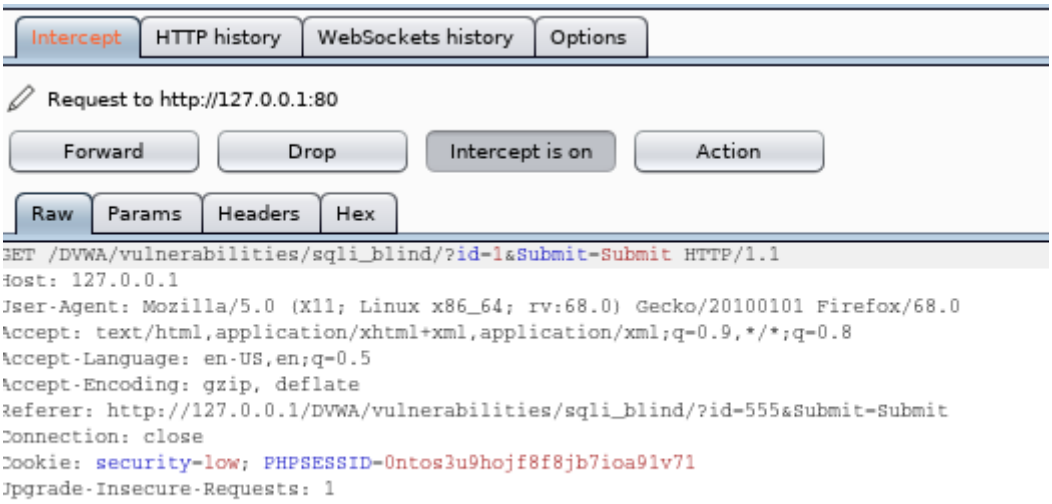
Odaberite payloads > payload options i pod simple lists uploadajte s Kali Linuxa set standardnih SQL injection napada koji se nalazi na /usr/share/wordlists/wfuzz/injection/SQL.txt.

Pokrenite napad, analizirajte rezultate, odaberite jedan uspješan napad i objasnite postupak.

SQLmap

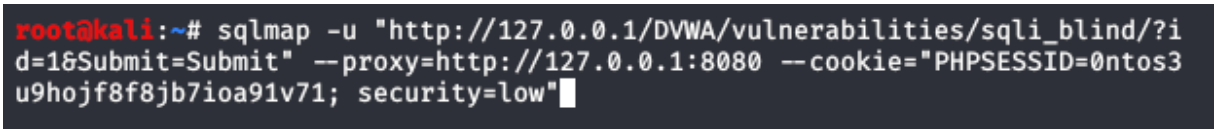
U ovom dijelu vježbe ćemo testirati SQLmap alat. Sve radite s ID 5.

Za početak, uhvatite GET request u BurpSuite-u (trbat će vam URL i PHPSESSID).



U terminalu utipkajte sqlmap --help što će vam pokazati opcije koje su dostupne u alatu.

Pokrenite naredbu (stavite vaše podatke ukoliko su primjenjivi):



Priložite screenshot. Objasnite rezultat rada naredbe.

Isprobajte istu naredbu s parametrom -dbs i još jednim parametrom po izboru. Priložite screenshot.

Pronađite tablice u bazi dvwa (opcija -D i --tables). Priložite screenshot.

Pronađite stupce u tablici users u bazi dvwa (opcija -D, opcija -T, opcija --columns). Priložite screenshot.

Na kraju uzmite dump tablice users (-D opcija za bazu, -T opcija za tablicu i opcija --dump).

Objasnite što ste dobili u dumpu i iskoristite opciju da se provede dictionary napad na password hasheve. Objasnite što je dictionary napad. Objasnite što ste dobili nakon napada i priložite screenshot. Probajte ući u DVWA s username-om i passwordom koji ste dobili za usera pod ID 5. Kao dokaz screenshotajte pri dnu DVWA aplikacije user info.