



# Module 15

# Hacking Web Application

**Ansh Bhawnani**

# Web Application Concepts

**Module 15**



# 1. Introduction

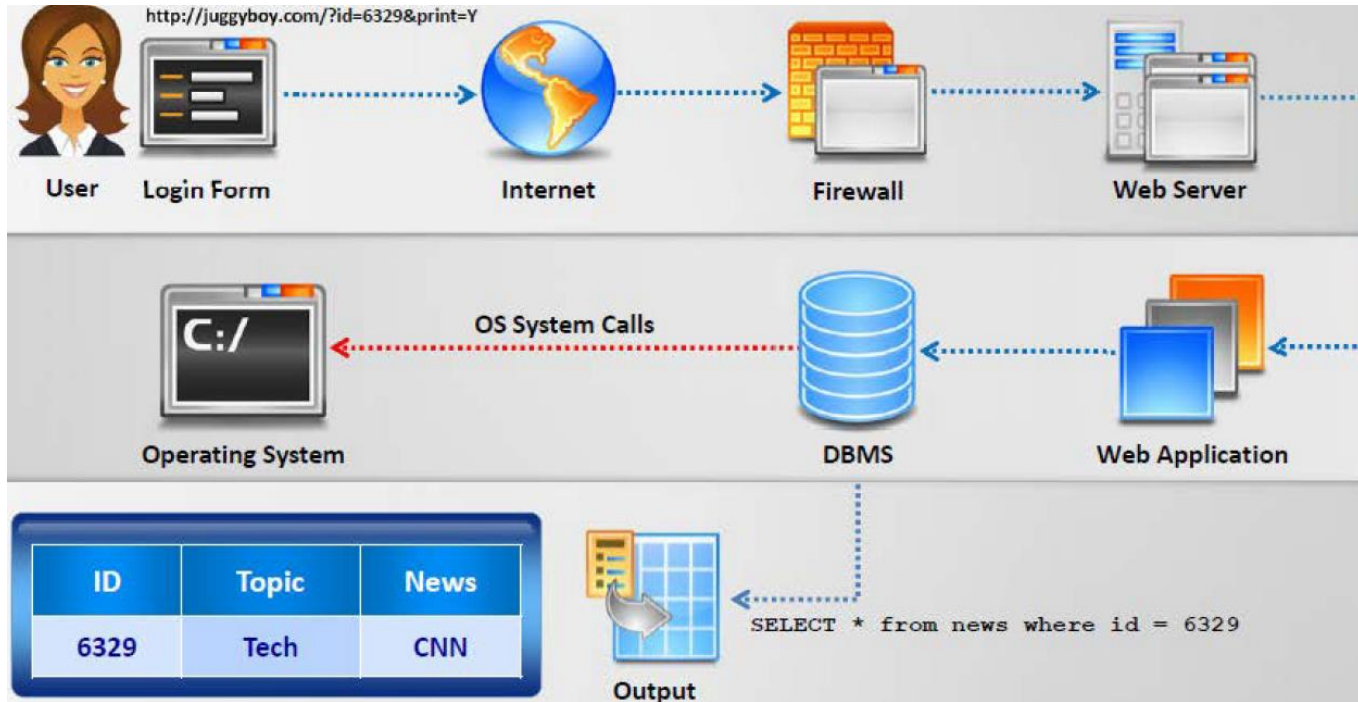


## Web Application Concepts

- Web applications provide an **interface** between **end users** and **web servers** through a **set of web pages** that are **generated** at the server end or contain **script code** to be **executed dynamically** within the client web browser.
- Though web application enforce certain **security policies**, they are **vulnerable** to various attacks such as SQL injection, cross-site scripting, session hijacking, etc.
- Web technologies such as **Web 2.0** provide **more attack surface** for web application exploitation.
- Web applications and Web 2.0 technologies are **invariably** used to support **critical business functions** such as **CRM, SCM**, etc. and improve **business efficiency**.



# Web Application Concepts

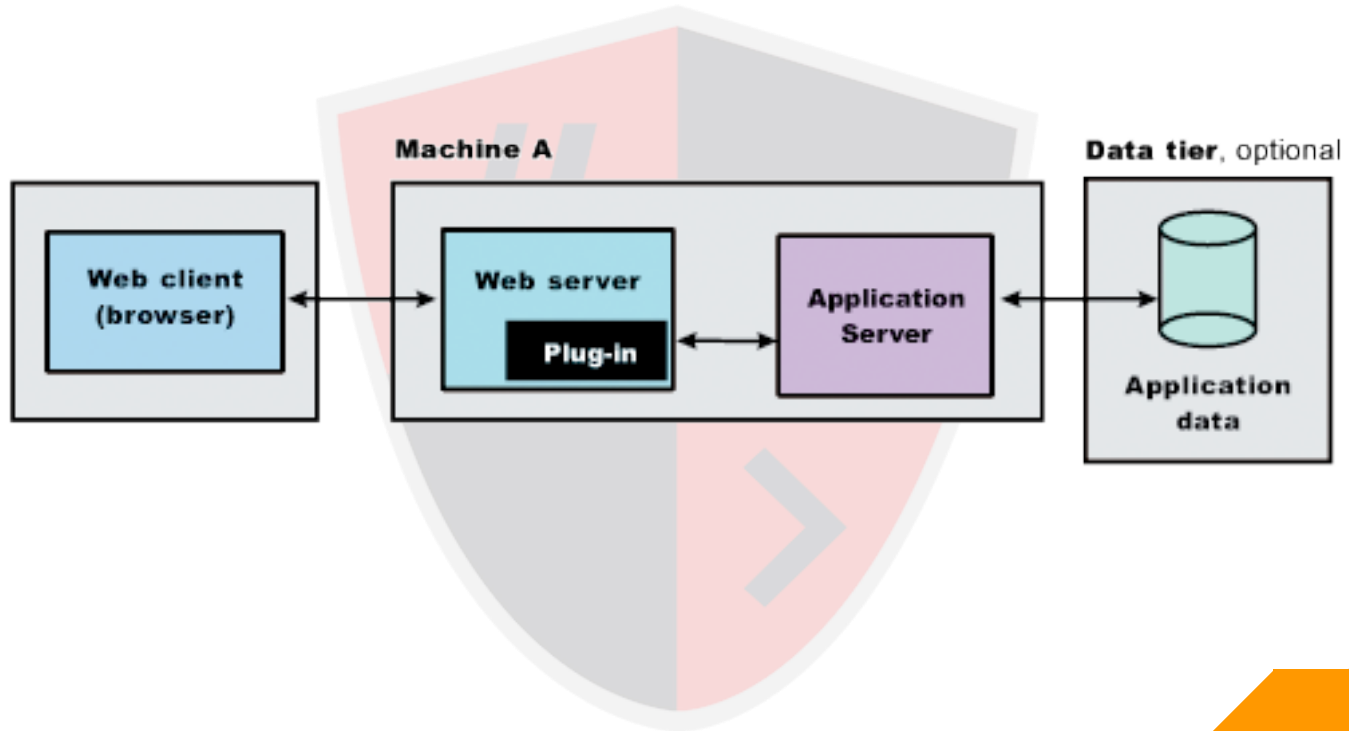




## 2. Web Application Components



# Web Application Concepts



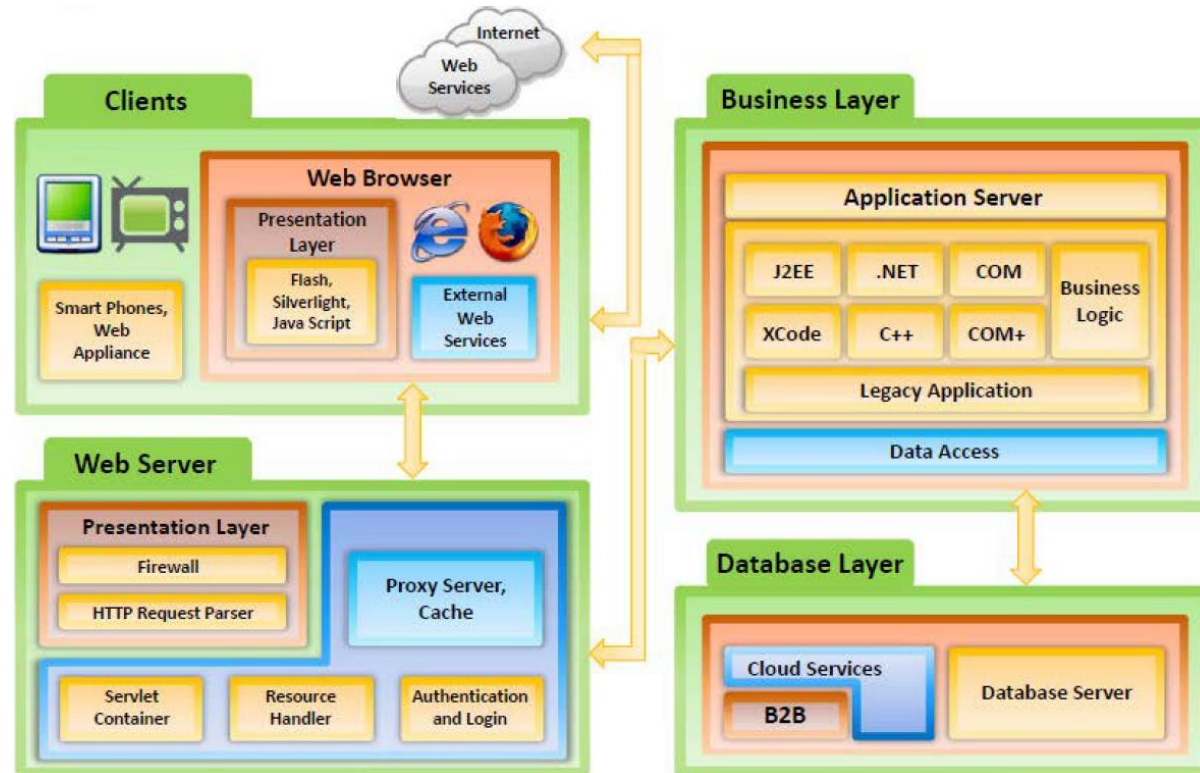


# 3. Web Application Architecture





# Web Application Concepts





# 4. Web 2.0 Applications



# Web Application Concepts

- Web 2.0 refers to a generation of Web applications that provide an **infrastructure** for more **dynamic user participation**, social **interaction** and **collaboration**.
- It offers various features such as:
  - ▷ **Interoperability:**
    - ▷ Advanced gaming
    - ▷ Dynamic as opposed to static site content
  - ▷ **User-centered Design:**
    - ▷ Social networking sites (Flickr, Facebook, del.cio.us)
    - ▷ Wikis and other collaborative applications
    - ▷ Google Base and other free web services (Google Maps)



# Web Application Concepts

## ■ Collaboration on the Web:

- ▶ Online office software (Google Docs and Microsoft Light)
- ▶ Interactive encyclopedias and dictionaries
- ▶ Cloud computing websites such as Amazon.com

## ■ Interactive Data Sharing:

- ▶ Frameworks (Yahoo! UI Library, jQuery)
- ▶ Mobile application (iPhone)
- ▶ New technologies like AJAX (Gmail, YouTube)
- ▶ Blogs (Wordpress)



# 5. Vulnerability Stack



# Web Application Concepts

Stacks	Services
Level 7	<b>Custom Web Applications:</b> Business Logic Flaws Technical Vulnerabilities
Level 6	<b>Third Party Components:</b> Open Source / Commercial
Level 5	<b>Database:</b> Oracle / MySQL / MS SQL
Level 4	<b>Web Server:</b> Apache / Microsoft IIS
Level 3	<b>Operating System:</b> Windows / Linux / OS X
Level 2	<b>Network:</b> Router / Switch
Level 1	<b>Security:</b> IPS / IDS



# Web Application Attack Methodology

**Module 15**



# 1. Footprint Web Infrastructure





## Web Application Attack Methodology

- Web infrastructure footprinting is the first step in web application hacking; it helps attackers to **select victims** and **identify vulnerable** web applications.
- Server Discovery:** Discover the **physical** servers that **hosts** web application.
- Service Discovery:** Discover the **services running** on web servers that can be exploited as **attack paths** for web app hacking.
- Server Identification:** Grab **server banners** to identify the **make** and **version** of the web server **software**.
- Hidden Content Discovery:** Extract **content** and **functionality** that is **not directly linked** or **reachable** from the **main visible** content.



# Web Application Attack Methodology

## Server Discovery

- ▶ Server discovery gives information about the **location** of servers and ensures that the **target** server is **alive** on Internet.
- ▶ **Whois Lookup**: Whois lookup utility gives information about the **IP address** of web server and **DNS names**
- ▶ **DNS Interrogation**: DNS interrogation provides information about the **location** and **type of servers**
- ▶ **Port Scanning**: Port Scanning attempts to connect to a particular set of **TCP or UDP ports** to find out the **service** that **exists** on the server.
  - ▶ Scan the target web server to identify common ports that web servers use for different services.



# Web Application Attack Methodology

- 
- ▶ Tools used for service discovery:
    - ▶ **Nmap**
    - ▶ **NetScan Tools Pro**
    - ▶ **Sandcat Browser**
  - ▶ Identified services act as attack paths for web application hacking.



# Web Application Attack Methodology

Port	Typical HTTP Services
80	World Wide Web standard port
81	Alternate WWW
88	Kerberos
443	SSL (https)
900	IBM Websphere administration client
2301	Compaq Insight Manager
2381	Compaq Insight Manager over SSL
4242	Microsoft Application Center Remote management
7001	BEA Weblogic
7002	BEA Weblogic over SSL
7070	Sun Java Web Server over SSL
8000	Alternate Web server, or Web cache
8001	Alternate Web server or management
8005	Apache Tomcat
9090	Sun Java Web Server admin module
10000	Netscape Administrator interface



# Web Application Attack Methodology

## Server Identification/Banner Grabbing

- ▶ Analyze the server **response header** field to **identify** the **make, model** and **version** of the web server software.
- ▶ **Syntax:** C:\telnet Website URL or IP address 80
- ▶ Run command **s\_client** -host [target website] -port 443
  - ▶ openssl.exe
- ▶ Type **GET / HTTP/1.0** to get the server information
- ▶ **Banner Grabbing Tools:**
  - ▶ Telnet, Netcat, ID Serve, Netcraft



# Web Application Attack Methodology

## ■ Detecting Proxies:

- ▶ Determine whether your target site is **routing** your **requests through** a proxy servers.
- ▶ Proxy servers generally **add certain headers** in the **response** header field.
- ▶ Use **TRACE** method of **HTTP/1.1** to identify the changes the proxy server made to the request.



# Web Application Attack Methodology

## ■ Detecting Web App Firewall:

- ▶ Web Application Firewall (WAF) prevents web application attack by **analyzing HTTP traffic**.
- ▶ Determine whether your target site is running web app firewall in **front** of an web **application**.
- ▶ **Check the cookies response** of your request because most of the **WAFs** **add their own cookie** in the response.
- ▶ Use **WAF detection tools** such as **WAFW00F** to find which WAF is running in front of application.



# Web Application Attack Methodology

## Hidden Content Discovery

- ▶ Discover the hidden content and functionality that is **not reachable** from the **main visible** content to **exploit user privileges** within the application.
- ▶ It allows an attacker to **recover backup copies** of **live files**, **configuration files** and **log files** containing sensitive data, **backup archives** containing **snapshots** of files within the web **root**, **new functionality** which is **not linked** to the main application, etc.





# Web Application Attack Methodology

## ■ Web Spidering:

- ▶ Web spiders automatically discover the hidden content and functionality by parsing HTML from the client-side JavaScript requests and responses.
- ▶ **Web Spidering Tools:**
  - ▶ OWASP Zed Attack Proxy
  - ▶ Burp Suite
  - ▶ WebScarab



# Web Application Attack Methodology

## Attacker-Directed Spidering:

- ▶ Attacker **accesses** all of the application's **functionality** and uses an **intercepting proxy** to **monitor** all **requests** and **responses**.
- ▶ The intercepting proxy **parses** all of the application's **responses** and **reports the content** and functionality it **discovers**.
- ▶ **Tool:** OWASP Zed Attack Proxy, Burpsuite

## Brute-Forcing:

- ▶ Use automation tools such as Burp Suite to make **huge numbers** of **requests** to the **web server** in order to **guess** the **names** or **identifiers** of **hidden content** and functionality.



# Web Application Attack Methodology

## Web Spidering Using Burp Suite

- ▶ Configure your web browser to use Burp as a local proxy.
- ▶ Access the entire target application visiting every single link/URL possible, and submit all the application forms available.
- ▶ Browse the target application with JavaScript enabled and disabled, and with cookies enabled and disabled.
- ▶ Check the site map generated by the Burp proxy, and identify any hidden application content or functions.
- ▶ Continue these steps recursively until no further content or functionality is identified.



## 2. Attacking Web Servers



# Web Application Attack Methodology

- Scan the server for **known vulnerabilities** using any web server vulnerability scanner.
- Launch web server **attack** to exploit identified vulnerabilities.
- **Tools used:**
  - ▷ UrlScan
  - ▷ Nikto
  - ▷ Nessus
  - ▷ Acunetix Web Vulnerability
  - ▷ WebInspect
- Launch **Denial-of-Service** (DoS) against web server.
  - ▷ DoSHTTP, Hping, Loci and Xoic, SYN Flooding, Slowloris, DRDoS.



# 3. Analyze Web Application



# Web Application Attack Methodology

- Analyze the **active application's functionality** and **technologies** in order to identify the **attack surfaces** that it exposes.
- Identify **Entry Points for User Input**: **Review** the generated **HTTP request** to identify the user input entry points.
- Identify **Server-Side Functionality**: **Observe** the **applications revealed** to the client to identify the server-side structure and functionality. [Common Gateway Interface (CGI)]
- Identify **Server-Side Technologies**: **Fingerprint** the **technologies active** on the server using various fingerprint techniques such as **HTTP fingerprinting**.
  - ▶ ASP, ASP.NET, ColdFusion, JSP, PHP, Python, and Ruby on Rails.
- **Map the Attack Surface**: **Identify** the various attack surfaces **uncovered** by the applications and the **vulnerabilities** that are **associated with each one**.



# Web Application Attack Methodology

## Identify Entry Points for User Input

- ▶ Examine **URL**, **HTTP Header**, **query string** parameters, **POST** data, and **cookies** to determine all user input fields.
- ▶ Identify HTTP **header** parameters that can be **processed** by the application as user inputs such as **User-Agent**, **Referer**, **Accept**, **Accept-Language**, and **Host** headers.
- ▶ **Determine** **URL encoding** techniques and other **encryption measures** implemented to secure the web traffic such as **SSL**.
- ▶ Tools used:
  - ▶ Burp Suite, HttPrint, WebScarab, OWASP Zed Attack Proxy





# Web Application Attack Methodology

## Identify Server-Side Technologies

- ▶ Perform a detailed server fingerprinting, analyze HTTP headers and HTML source code to identify server side technologies.
- ▶ Examine URLs for file extensions, directories, and other identification information.
- ▶ Examine the error page messages.
- ▶ Examine session tokens:
  - ▶ **JSESSIONID** - Java
  - ▶ **ASPSESSIONID** - IIS server
  - ▶ **ASP.NET\_SessionId** - ASP.NET
  - ▶ **PHPSESSID** - PHP



# Web Application Attack Methodology

## Identify Server-Side Functionality

- ▶ Examine page source and URLs and make an educated guess to determine the internal structure and functionality of web applications.
- ▶ **Tools used:**
  - ▶ GUN Wget, Teleport Pro, BlackWidow
- ▶ **Examine URL:**
  - ▶ `https://www.juggyboy.com/customers.aspx?name=existing%20clients&isActive=O&startDate=20%2F11%2F2010&endDate=20%2F05%2F2011&showBy=name`
  - ▶ **https:** SSL
  - ▶ **aspx:** ASPX | Platform



# Web Application Attack Methodology

Information	Attack	Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack	Injection Attack	Privilege Escalation, Access Controls
Database Interaction	SQL Injection, Data Leakage	Cleartext Communication	Data Theft, Session Hijacking
File Upload and Download	Directory Traversal	Error Message	Information Leakage
Display of User-Supplied Data	Cross-Site Scripting	Email Interaction	Email Injection
Dynamic Redirects	Redirection, Header Injection	Application Codes	Buffer Overflows
Login	Username Enumeration, Password Brute-Force	Third-Party Application	Known Vulnerabilities Exploitation
Session State	Session Hijacking, Session Fixation	Web Server Software	Known Vulnerabilities Exploitation



# **4. Attack Authentication Mechanism**



# Web Application Attack Methodology

■ Attackers can exploit design and implementation flaws in web applications, such as failure to check password strength or insecure transportation of credentials, to bypass authentication mechanisms.

■ **User Name Enumeration:**

- ▶ Verbose failure messages
- ▶ Predictable user names

■ **Cookie Exploitation:**

- ▶ Cookie poisoning
- ▶ Cookie sniffing
- ▶ Cookie replay



# Web Application Attack Methodology

## ■ Session Attacks:

- ▷ Session prediction
- ▷ Session brute-forcing
- ▷ Session poisoning

## ■ Password Attacks:

- ▷ Password functionality exploits
- ▷ Password guessing
- ▷ Brute-force attack



# Web Application Attack Methodology

## User Name Enumeration

- If **login error** states which **part** of the user name and password is **not correct**, **guess the users** of the application using the **trial-and-error** method.
- Some applications **automatically generate** account **user names** based on a **sequence** (such as **user101**, **user102**, etc.), and attackers can **determine the sequence** and **enumerate** valid user names.
- **Note:** User name **enumeration** from verbose error messages will **fail** if the application **implements account lockout** policy i.e., locks account after a certain number of failed login attempt.



# Web Application Attack Methodology

## ■ Password Functionality Exploits

- ▷ **Password Changing:**
  - ▷ Determine password change functionality within the application by **spidering** the application or **creating a login account**.
  - ▷ Try **random strings** for 'Old Password', 'New Password', and 'Confirm the New Password' fields and **analyze errors** to identify vulnerabilities in password change functionality.





# Web Application Attack Methodology

## Password Recovery:

- 'Forgot Password' features generally present a **challenge** to the user; if the number of **attempts** is **not limited**, attacker can **guess the challenge** answer successfully with the help of **social engineering**.
- Applications may also send a **unique recovery URL** or existing password to an **email address** specified by the attacker if the **challenge is solved**.

## "Remember Me" Exploit:

- "Remember Me" functions are **implemented** using a simple **persistent cookie**, such as RememberUser=jason or a persistent **session identifier** such as RememberUser=ABY112010.
- Attackers can use an **enumerated user name** or **predict the session** identifier to bypass authentication mechanisms.



# Web Application Attack Methodology

## ■ Cookie Exploitation: Cookie Poisoning

- ▶ If the **cookie contains passwords** or **session** identifiers, attackers can **steal the cookie** using techniques such as **script injection** and **eavesdropping**.
- ▶ Attackers then **replay then cookie** with the same or altered passwords or session identifiers to bypass web application authentication.
- ▶ Attackers can **trap cookies** using **tools** such as OWASP Zed Attack Proxy, Burp Suite, etc.



# 5. Attack Authorization Schemes



# Web Application Attack Methodology

## Authorization Attack

- ▶ Attackers **manipulate** the HTTP requests to **subvert** the application authorization schemes by **modifying input fields** that relate to user ID, user name, access group, cost, filenames, file identifiers, etc.
- ▶ Attackers first **access** web application using **low privileged account** and then **escalate privilege** to access **protected** resources.
- ▶ Attackers use sources such as the following to perform authorization attacks:
  - ▶ **Parameter Tampering, POST Data, Uniform Resource Identifier, HTTP Headers, Cookies, Hidden Tags**



# Web Application Attack Methodology

## HTTP Request Tampering

### ▶ Query String Tampering:

- ▶ If the **query string is visible** in the **address bar** on the browser, the attacker can **easily change** the string parameter to bypass authorization mechanisms.
- ▶ `http://www.juggyboy.com/mail.aspx?mailbox=john&company=acme%20com`
- ▶ `https://juggyshop.com/books/download/852741369.pdf`
- ▶ `https://juggybank.com/login/home.jsp?admin=true`
- ▶ Attackers can use **web spidering tools** such as Burp Suite to scan the web app for **POST** parameters.



# Web Application Attack Methodology

## HTTP Headers:

- ▶ If the application uses the **Referer header** for **making access control** decisions, attackers can **modify it to** access protected application functionalities.

GET http://juggyboy:8180/Application/Download?ItemID=201 HTTP/1.1

Host: janaina:8180

...

**Referer:** http://juggyboy:8180/Application/Download?Admin=False

- ▶ ItemID=201 is **not accessible as Admin** parameter is set to **false**, attacker can **change it to true** and access protected items.



# Web Application Attack Methodology

## Cookie Parameter Tampering

- ▶ In the first step, the attacker **collects some cookies** set by the web application and **analyzes them to determine** the cookie **generation mechanism**.
- ▶ The attacker then **traps cookies** set by the web application, **tampers** with its **parameters** using tools, such as OWASP Zed Attack Proxy, and **replay** to the application.



# 6. Attack Session Management Mechanism





# Web Application Attack Methodology

## Session Management Attack

- ▶ Attackers **break** an application's session management mechanism to bypass the authentication controls and **impersonate privileged** application users.
- ▶ **Session Token Generation:**
  - ▶ Session Tokens **Prediction**
  - ▶ Session Tokens **Tampering**
- ▶ **Session Tokens Handling:**
  - ▶ **Man-In-The-Middle** Attack
  - ▶ Session **Replay**
  - ▶ Session **Hijacking**



# Web Application Attack Methodology

## Session Token Generation Mechanism

### ▶ Weak Encoding Example:

- ▶ `https://www.juggyboy.com/checkout?SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%3B%64%61%74%65%3D%32%33%2F%31%31%2F%32%30%31%30`
- ▶ When **hex-encoding of an ASCII string** `user=jason;app=admin;date=23/11/2010`, the attacker can **predict another session token** by just **changing date** and **use** it for **another transaction** with server.



# Web Application Attack Methodology

## Session Token Prediction:

- ▶ Attackers obtain valid session token by sniffing the traffic or legitimately logging into application and analyzing it for encoding (hex-encoding, Base64) or any pattern.
- ▶ If any meaning can be reverse engineered from the sample of session tokens, attackers attempt to guess the tokens recently issued to other application users.
- ▶ Attackers then make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token.



# Web Application Attack Methodology

## ■ Session Token Sniffing

- ▶ Attackers sniff the application traffic using a sniffing tool such as **Wireshark** or an **intercepting proxy** such as **Burp**. If HTTP cookies are being used as the transmission mechanism for session tokens and the **secure flag is not set**, attackers can **replay** the cookie to gain unauthorized access to application.
- ▶ Attacker can use session cookies to perform **session hijacking**, session **replay**, and **Man-in-the-Middle** attacks.



# 7. Performing Various Attacks



# Web Application Attack Methodology

## Injection Attacks/Input Validation Attacks (?)

- ▶ In injection attacks, attackers supply **crafted malicious input** that is **syntactically correct** according to the **interpreted language** being used in order to **break application's normal** intended functionality.
- ▶ **Web Scripts Injection:** If user input is used into **dynamically executed code**, enter **crafted input** that **breaks** the **intended data context** and **executes commands** on the server.
- ▶ **OS Commands Injection:** Exploit operating systems by entering **malicious codes** in input fields if applications utilize user input in a **system-level command**.
- ▶ **SMTP Injection:** Injection **arbitrary SMTP commands** into application and SMTP **server conversation** to generate **large volumes of spam** email.



## Web Application Attack Methodology

- ▶ **SQL Injection:** Enter a series of **malicious SQL queries** into input fields to directly **manipulate the database**.
- ▶ **LDAP Injection:** Take advantage of **non-validated web application** input vulnerabilities to **pass LDAP filters** to obtain **direct access to databases**.
- ▶ **XPath Injection:** Enter malicious strings in input fields in order to **manipulate the XPath query** so that it **interferes** with the application's **logic**.
- ▶ **Buffer Overflow:** Injections **large amount of bogus data** beyond the **capacity** of the **input field**.
- ▶ **Canonicalization:** Manipulate variables that **reference files with "dot-dot-slash (../)"** to access restricted directories in the application.



# Web Application Attack Methodology

## Attack Data Connectivity (?)

- ▶ Database **connection strings** are used to **connect applications** to **database engines**.
- ▶ Example of a common connection string used to connect to a Microsoft SQL Server database: "Data Source=Server, Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;"
- ▶ Database connectivity attacks exploit the **way applications connect** to the database **instead of abusing database queries**.
- ▶ **Data Connectivity Attacks:**
  - ▶ Connection String Injection
  - ▶ Connection Pool DoS





# Web Application Attack Methodology

## Connection String Injection

- ▶ In a **delegated** authentication environment, the attacker **injects** parameters in a **connection string** by **appending** them **with the semicolon (;)** character.
- ▶ A connection string injection attack can occur when a **dynamic string concatenation** is used to **build connection** strings based on user input.
- ▶ **Before Injection:**
  - ▶ "Data Source=Server, Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;"



# Web Application Attack Methodology

## Connection String Injection

- ▶ **After Injection:**
  - ▶ "Data Source=Server, Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;Encryption=off"
- ▶ When the connection string is **populated**, the **Encryption value** will be **added** to the **previously configured** set of **parameters**.



# Web Application Attack Methodology

## Connection String Parameter Pollution (CSPP) Attacks

- ▶ In CSPP attacks, attackers **overwrite parameter values** in the connection string.
- ▶ **Hash Stealing:**
  - ▶ Attacker **replaces the value** of **Data Source** parameter with that of a **Rogue Microsoft SQL Server** connected to the Internet **running a sniffer**.
  - ▶ `Data source = SQL2005; initial catalog = db1; integrated security=no; user id =;Data Source=Rogue Server;Password=;Integrated Security=true;`
  - ▶ Attacker will then **sniff Windows credentials** (password **hashes**) when the application **tries to connect** to **Rogue\_Server** with the Windows credentials it's running on.



# Web Application Attack Methodology

## Port Scanning:

- ▶ Attacker tries to **connect to different ports** by **changing the value** and **seeing the error messages** obtained.
- ▶ Data source = SQL2005; initial catalog = db1; integrated security=no; user id =;Data Source=Target Server, Target Port=443;Password=;Integrated Security=true;

## Hijacking Web Credentials:

- ▶ Attacker tries to **connect to the database** by using the Web Application System **account instead** of a **user-provided set** of credentials.
- ▶ Data source = SQL2005; initial catalog = db1; integrated security=no; user id =;Data Source=Target Server, Target Port;Password=;Integrated Security=true;



# Web Application Attack Methodology

## Connection Pool DoS

- ▶ Attacker examines the connection pooling settings of the application, constructs a large malicious SQL query, and runs multiple queries simultaneously to consume all connections in the connection pool, causing database queries to fail for legitimate users.
- ▶ **Example:** By default in ASP.NET, the maximum allowed connections in the pool is 100 and timeout is 30 seconds.
- ▶ Thus, an attacker can run 100 multiple queries with 30+ seconds execution time within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application.



# 8. Attack Web App Client



# Web Application Attack Methodology

- Attackers interact with the server-side applications in **unexpected ways** in order to perform **malicious actions against the end users** and access unauthorized data.
- **Cross-Site Scripting**: An attacker bypasses the client's security mechanism and obtains access privileges, and then **injects malicious scripts** into the **web pages** of a website. These malicious scripts can even **rewrite the HTML content** of the website.
- **HTTP Header Injection**: Attackers **split an HTTP response** into **multiple responses** by **injecting a malicious response** in an HTTP header. By doing so, attackers can **deface websites**, **poison the cache**, and trigger cross-site scripting.
- **Request Forgery Attack**: In a request forgery attack, attackers **exploit the trust** of a **website** or web application **on a user's browser**. The attack works by **including a link** on a page, which **takes the user to an authenticated website**.



## Web Application Attack Methodology

- **Privacy Attacks:** A privacy attack is tracking performed with the help of a remote site by employing a leaked persistent browser state.
- **Redirection Attacks:** Attackers develop codes and links that resemble a legitimate site that a user wants to visit; however, in so doing, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.
- **Frame Injection:** When scripts do not validate their input, attackers inject codes through frames. This affects all the browsers and scripts, which do not validate untrusted input. These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.





## Web Application Attack Methodology

- **Session Fixation:** Session fixation helps attackers **hijack valid user sessions**. They **authenticate themselves** using a **known session ID**, and then **use the already known** session ID to **hijack a user-validated** session. Thus, attackers **trick** the users into accessing a genuine web server using an existing session ID value.
- **ActiveX Attacks:** Attackers **lure victims** via **email** or via a **link** that attackers have **constructed in such a way** that **loopholes** of **remote execute code** become accessible, allowing the attackers to obtain access privileges equal to that of an authorized user.



# 9. Attack Web Services



## Web Application Attack Methodology

- Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks.
- Various types of attacks used to attack web services are:
  - ▷ SOAP Injection
  - ▷ XML Injection
  - ▷ WSDL Probing Attacks
  - ▷ Information Leakage
  - ▷ Application Logic Attacks
  - ▷ Database Attacks



# Web Application Attack Methodology

## ■ Probing Attacks

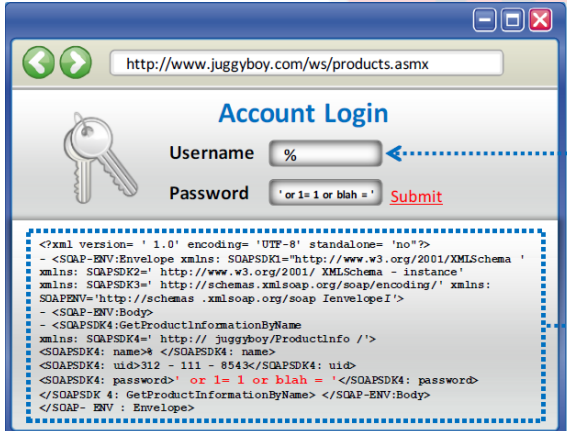
- The attacker **traps** the **WSDL document** from web service traffic and **analyzes** it to determine the **purpose of the application**, **functional break down**, **entry points**, and **message types**.
- Attacker then **creates a set of valid requests** by selecting a set of operations, and **formulating the request messages according** to the **rules** of the **XML Schema** that can be submitted to the web service.
- Attacker uses these requests to **include malicious contents** in **SOAP requests** and analyzes errors to **gain a deeper understanding** of potential security **weaknesses**.



# Web Application Attack Methodology

## SOAP Injection

- Attacker injects malicious query strings in the user input field to bypass web services authentication mechanisms and access backend databases.
- This attack works similarly to SQL Injection attacks.



Account Login

Username

Password

```
<?xml version=' 1.0' encoding=' UTF-8' standalone='no'?>
- <SOAP-ENV:Envelope xmlns: SOAPSDK1='http://www.w3.org/2001/XMLSchema '
xmlns: SOAPSDK2=' http://www.w3.org/2001/ XMLSchema - instance'
xmlns: SOAPSDK3=' http://schemas.xmlsoap.org/soap/encoding/' xmlns:
SOAPENV='http://schemas.xmlsoap.org/soap/EnvelopeI'>
- <SOAP-ENV:Body>
- <SOAPSDK4:GetProductInformationByName
xmlns: SOAPSDK4=' http:// juggyboy/ProductInfo /'>
<SOAPSDK4: name>4 </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password> ' or 1=1 or blah = ' </SOAPSDK4: password>
</SOAPSDK 4: GetProductInformationByName> </SOAP-ENV:Body>
</SOAP- ENV : Envelope>
```

### Server Response

```
<?xml version="1.0" encoding="utf-8" ?>
- <soap: Envelope xmlns: soap='http://schemas
.xmlsoap.org/soap/envelope/'
xmlns: xsi ='http://www.w3 .org/2001/XMLSchema-
instance'
xmlns: xsd='http://www.w3 .org/2001/XMLSchema'>
- <soap:Body>
- <GetProductInformationByNameResponse
xmlns="http://juggyboy/ProductInfo/">
- <GetProductInformationByNameResult>
<productid> 25 </productid>
<product Name >Painting101</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>
```



# Web Application Attack Methodology

## XML Injection (?)

- Attackers inject XML data and tags into user input fields to manipulate XML schema or populate XML database with bogus entries.
- XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.

Account Login

Username

Password

E-mail  [Submit](#)

mark@certifiedhacker.com</mail> </user> <user> <username>jason</username> <password>attck</password> <userid>105</userid> <mail>jason@juggyboy.com</mail> </user>

### Server Side Code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>101</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Mark</username>
    <password>12345</password>
    <userid>102</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>jason</username>
    <password>attck</password>
    <userid>105</userid>
    <mail>jason@juggyboy.com</mail>
  </user>
</users>
```

Creates new  
user account  
on the server



# Web Application Attack Methodology

## Parsing Attacks

- Parsing attacks exploit vulnerabilities and weaknesses in the **processing capabilities** of the **XML parser** to **create a denial-of-service** attack or **generate logical errors** in web service request processing.
- **Recursive Payloads**: Attacker queries for web services with a **grammatically correct SOAP** document that contains **infinite processing loops** resulting in **exhaustion** of **XML parser** and **CPU resources**.
- **Oversize Payloads**: Attackers send a payload that is **excessively large** to **consume all systems resources** rendering web services inaccessible to other legitimate users.



# Web Application Attack Methodology

## SoapUI and XMLSpy

### ▷ SoapUI:

- ▷ SoapUI is a **web service testing tool** which supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF, and JDBC.
- ▷ Attacker can use this tool to carry out **web services probing, SOAP injection, XML injection**, and web services **parsing attacks**.

### ▷ XMLSpy:

- ▷ Altova XMLSpy is the **XML editor** and **development environment** for **modeling, editing, transforming**, and **debugging XML-related technologies**.





# HACKING

Is an art, practised through a creative mind.

