Security Testing

protect the data and maintain functionality

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Security Testing is performed to reveal security flaws in the system in order to protect data and maintain functionality.

This tutorial explains the core concepts of Security Testing and related topics with simple and useful examples.

Audience

This tutorial has been prepared for beginners to help them understand the basics of security testing.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of software testing and its related concepts.

Copyright & Disclaimer

Table of Contents

1. Overview

Security testing is very important to keep the system protected from malicious activities on the web.

What is Security Testing?

Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended. Security testing does not guarantee complete security of the system, but it is important to include security testing as a part of the testing process.

Security testing takes the following six measures to provide a secured environment:

- **Confidentiality -** It protects against disclosure of information to unintended recipients.
- **Integrity -** It allows transferring accurate and correct desired information from senders to intended receivers.
- **Authentication -** It verifies and confirms the identity of the user.
- **Authorization -** It specifies access rights to the users and resources.
- **Availability -** It ensures readiness of the information on requirement.
- **Non-repudiation -** It ensures there is no denial from the sender or the receiver for having sent or received the message.

Example

Spotting a security flaw in a web-based application involves complex steps and creative thinking. At times, a simple test can expose the most severe security risk. You can try this very basic test on any web application:

1. Log into the web application using valid credentials.
2. Log out of the web application.
3. Click the BACK button of the browser.
4. Verify if you are asked to log in again or if you are able go back to the logged in page again.

2. Security Testing Process

Security testing can be seen as a controlled attack on the system, which uncovers security flaws in a realistic way. Its goal is to evaluate the current status of an IT system. It is also known as **penetration test** or more popularly as **ethical hacking**.

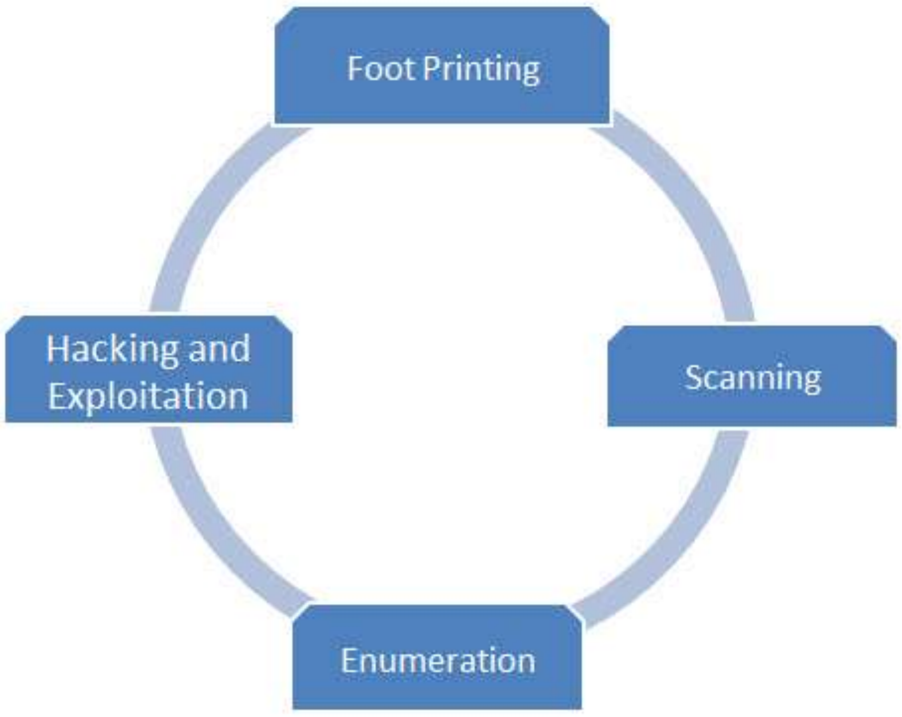
Penetration test is done in phases and here in this chapter, we will discuss the complete process. Proper documentation should be done in each phase so that all the steps necessary to reproduce the attack are available readily. The documentation also serves as the basis for the detailed report customers receive at the end of a penetration test.

Penetration Test – Workflow

Penetration test includes four major phases:

- Foot Printing
- Scanning
- Enumeration
- Exploitation

These four steps are re-iterated multiple times which goes hand in hand with the normal SDLC.



Footprinting

Footprinting is the process of gathering the blueprint of a particular system or a network and the devices that are attached to the network under consideration. It is the first step that a penetration tester uses to evaluate the security of a web application.

After footprinting, a penetration tester can understand the pulse of a hacker. It is good to understand the complete system before testing its modules.

Footprinting – Steps

- Information gathering
- Determining the range of the network
- Identifying active machines
- Identifying open ports and access points
- OS fingerprinting
- Fingerprinting services
- Mapping the network

Tools Used in Footprinting

Following are the common set of tools used in footprinting:

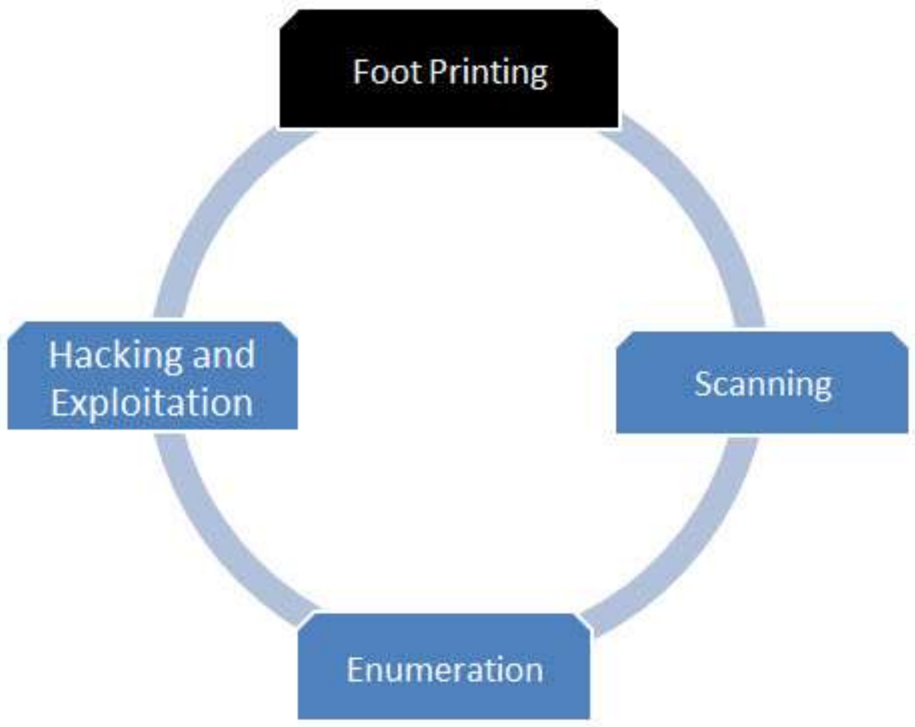
- Whois
- SmartWhois
- NsLookup
- Sam Spade

Other Techniques Used in Footprinting

Footprinting may also involve collecting information such as:

- Company contact names, email addresses, and phone numbers
- Company deals and other parties involved
- News on mergers and acquisitions
- Links to other company-related sites
- Company's privacy policies

Flow Diagram



Scanning

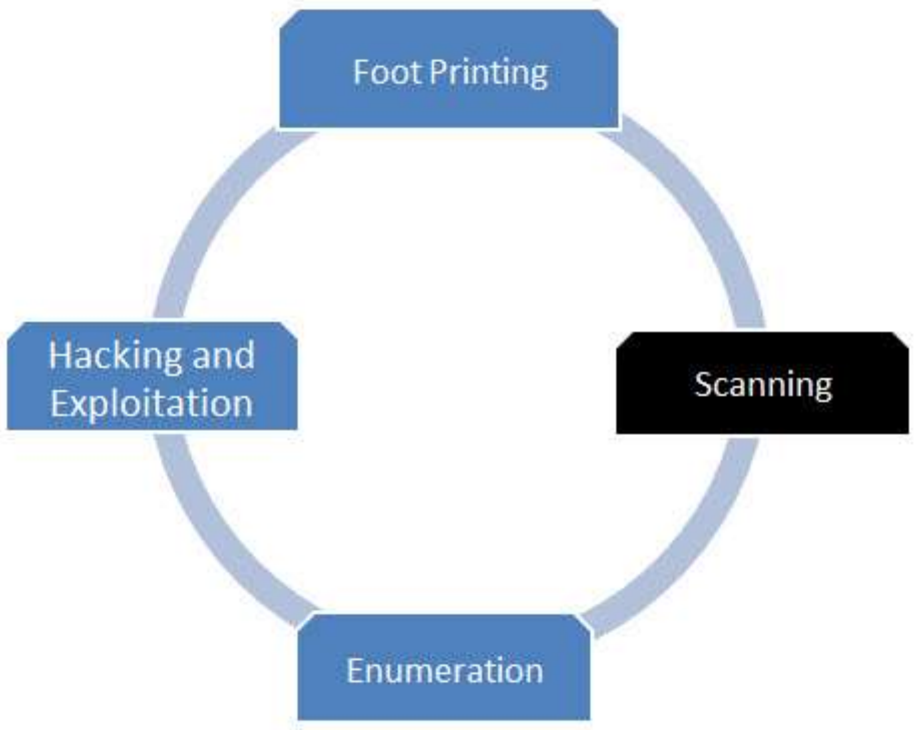
Scanning is the second step that is performed after footprinting. It involves scanning open ports, fingerprinting the operating system, and uncovering services on ports. The ultimate goal of scanning is to find open ports through external or internal network scanning, pinging machines, determining network ranges, and port scanning individual systems.

Tools Used in Scanning

Following are the common set of tools/resources used in Scanning:

- NMap
- Ping
- Traceroute
- Superscan
- Netcat
- NeoTrace

Flow Diagram



Enumeration

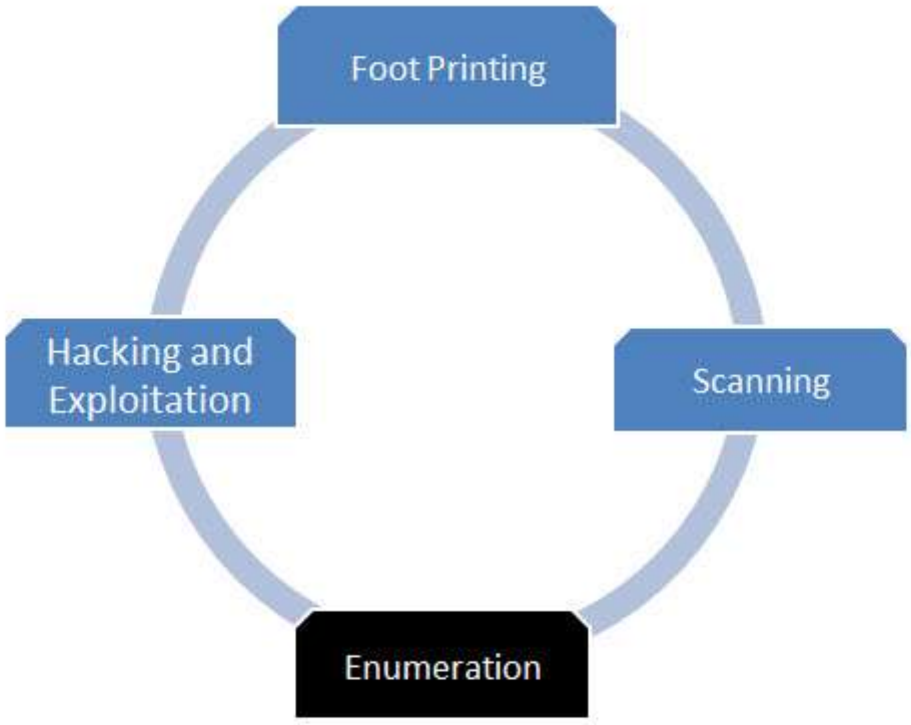
Enumeration is the next step after scanning. The goal of enumeration is to get a complete picture of the target. In this phase, a penetration tester tries to identify valid user accounts or poorly-protected shared resources using active connections to systems.

Techniques Used in Enumeration

Following are the common set of procedures used in Enumeration:

- Identifying vulnerable user accounts
- Obtaining Active Directory information
- Using snmputil for Simple Network Management Protocol enumeration
- Employing Windows DNS queries
- Establishing null sessions and connections

Flow Diagram



Exploitation

Exploitation is the last phase where a security tester actively exploits the security weaknesses present in the system under consideration. Once the attack is successful, it is possible to penetrate more systems in the domain, because the penetration testers then have the access to more potential targets that were not available before.

Techniques Used in Exploitation

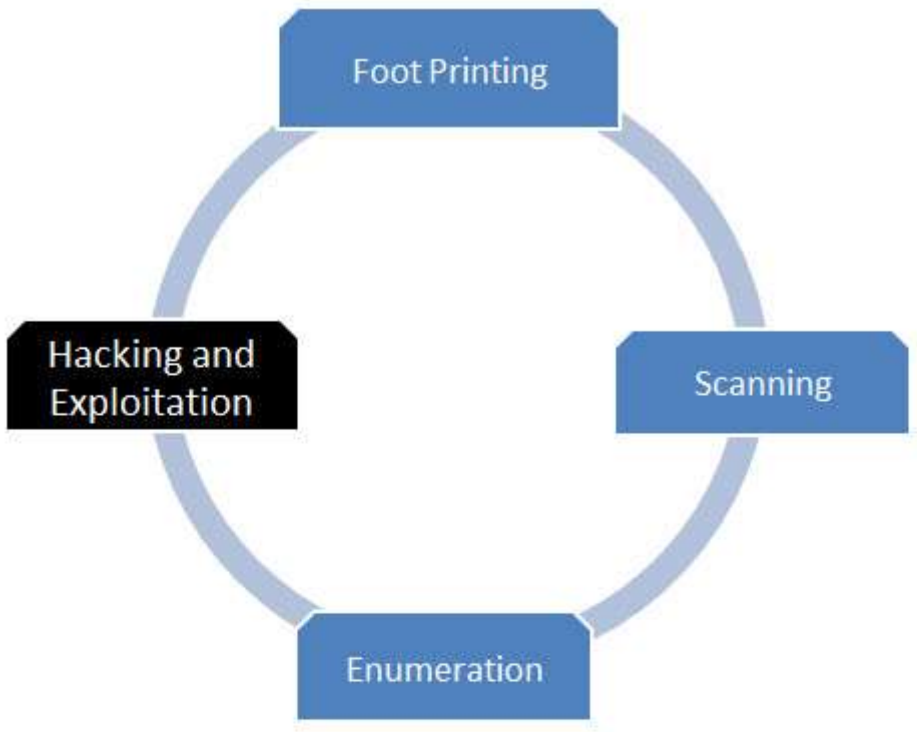
The types of exploitation are segregated into three different categories:

1. **Attack against WEB-SERVERS**
    - SQL Injection
    - Cross-site Scripting
    - Code Injection
    - Session Hijacking
    - Directory Traversal

2. **Attack against NETWORKS**
    - Man in the Middle Attack
    - Spoofing
    - Firewall Traversal
    - WLAN
    - ARP Poisoning

3. **Attack against SERVICES**
    - Buffer Overflows
    - Format Strings
    - Dos
    - Authentication flaws

Flow Diagram



3. Malicious Software

Malicious software (malware) is any software that gives partial to full control of the system to the attacker/malware creator.

Malwares

Various forms of malware are listed below:

- **Virus** – A virus is a program that creates copies of itself and inserts these copies into other computer programs, data files, or into the boot sector of the hard-disk. Upon successful replication, viruses cause harmful activity on infected hosts such as stealing hard-disk space or CPU time.

- **Worm** - A worm is a type of malware which leaves a copy of itself in the memory of each computer in its path.

- **Trojan** - Trojan is a non-self-replicating type of malware that contains malicious code, which upon execution results in loss or theft of data or possible system harm.

- **Adware –** Adware, also known as freeware or pitchware, is a free computer software that contains commercial advertisements of games, desktop toolbars, and utilities. It is a web-based application and it collects web browser data to target advertisements, especially pop-ups.

- **Spyware** - Spyware is infiltration software that anonymously monitors users which enables a hacker to obtain sensitive information from the user's computer. Spyware exploits users and application vulnerabilities that is quite often attached to free online software downloads or to links that are clicked by users.

- **Rootkit** - A rootkit is a software used by a hacker to gain admin level access to a computer/network which is installed through a stolen password or by exploiting a system vulnerability without the victim's knowledge.

Preventive Measures

The following measures can be taken to avoid presence of malware in a system:

- Ensure the operating system and applications are up to date with patches/updates.

- Never open strange e-mails, especially ones with attachments.

- When you download from the internet, always check what you install. Do not simply click OK to dismiss pop-up windows. Verify the publisher before you install application.

- Install anti-virus software.

- Ensure you scan and update the antivirus programs regularly.

- Install firewall.
- Always enable and use security features provided by browsers and applications.

Anti-Malware Software

The following software help remove the malwares from a system:

- Microsoft Security Essentials
- Microsoft Windows Defender
- AVG Internet Security
- Spybot - Search & Destroy
- Avast! Home Edition for personal use
- Panda Internet Security
- MacScan for Mac OS and Mac OS X

4. HTTP Protocol Basics

Understanding the protocol is very important to get a good grasp on security testing. You will be able to appreciate the importance of the protocol when we intercept the packet data between the webserver and the client.

HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extension of its request methods, error codes, and headers.

Basically, HTTP is a TCP/IP based communication protocol, which is used to deliver data such as HTML files, image files, query results etc. over the web. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' requested data are sent to the server, and how servers respond to these requests.

Basic Features

There are following three basic features which make HTTP a simple yet powerful protocol:

- **HTTP is connectionless:** The HTTP client, i.e., the browser initiates an HTTP request. After making a request, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send the response back.

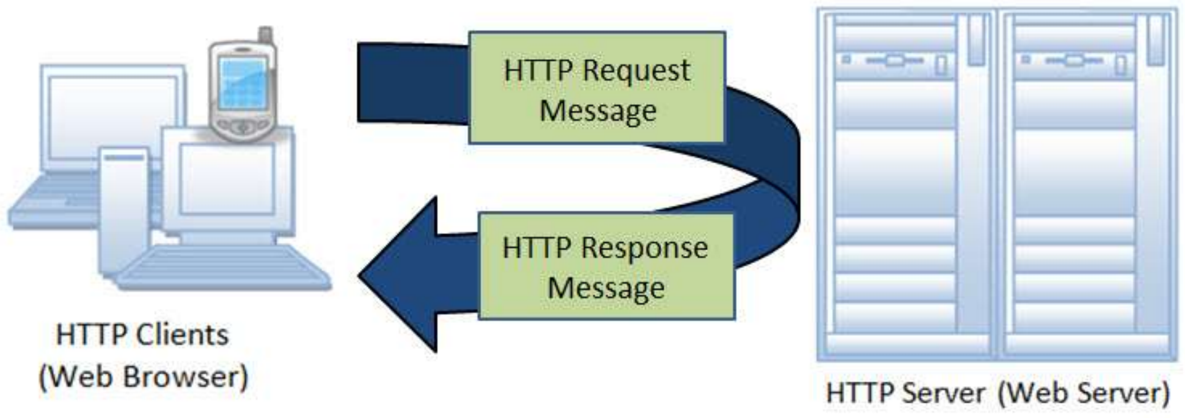
- **HTTP is media independent:** Any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. This is required for client as well as server to specify the content type using appropriate MIME-type.

- **HTTP is stateless:** HTTP is a connectionless and this is a direct result that HTTP is a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

HTTP/1.0 uses a new connection for each request/response exchange whereas HTTP/1.1 connection may be used for one or more request/response exchanges.

Architecture

The following diagram shows a very basic architecture of a web application and depicts where HTTP resides:



The HTTP protocol is a request/response protocol based on the client/server architecture where web browser, robots, and search engines etc. act as HTTP clients and the web server acts as a server.

- **Client -** The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

- **Server -** The HTTP server responds with a status line, including the protocol version of the message and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

HTTP – Disadvantages

- HTTP is not a completely secured protocol.

- HTTP uses port 80 as default port for communication.

- HTTP operates at the application Layer. It needs to create multiple connections for data transfer, which increases administration overheads.

- No encryption/digital certificates are required for using HTTP.

HTTP Parameters

We will discuss here a few important HTTP Protocol Parameters and their syntax that are required in constructing the request and response messages while writing HTTP client or server programs. We will cover the complete usage of these parameters in subsequent chapters while explaining the message structure for HTTP requests and responses.

HTTP Version

HTTP uses a **<major>.<minor>** numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number:

```
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Example

```
HTTP/1.0


or


HTTP/1.1
```

Uniform Resource Identifiers (URI)

URI is simply formatted, case-insensitive string containing name, location etc. to identify a resource. For example, a website name, a web service etc. A general syntax of URI used for HTTP is as follows:

```
URI = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Here, if the **port** is empty or not given, port 80 is assumed for HTTP and an empty **abs_path** is equivalent to an **abs_path** of "/". The characters other than those in the **reserved** and **unsafe** sets are equivalent to their ""%" HEX HEX" encoding.

Example

Following two URIs are equivalent:

```
http://abc.com:80/~smith/home.html

http://ABC.com/%7Esmith/home.html

http://ABC.com:/%7esmith/home.html
```

Date/Time Formats

All HTTP date/time stamps must be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036

Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

Character Sets

You use character set to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is US-ASCII.

Example

The following character sets are valid:

```
US-ASCII


or


ISO-8859-1


or


ISO-8859-7
```

Content Encodings

Content encoding values indicate that an encoding algorithm is used to encode the content before passing it over the network. Content encodings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields.

Example

The following are valid encoding schemes:

```
Accept-encoding: gzip


or


Accept-encoding: compress


or


Accept-encoding: deflate
```

Media Types

HTTP uses Internet Media Types in the **Content-Type** and **Accept** header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority ((IANA). The following general syntax specifies media type:

```
media-type     = type "/" subtype *( ";" parameter )
```

The type, subtype, and parameter attribute names are case- insensitive.

Example

```
Accept: image/gif
```

Language Tags

HTTP uses language tags within the **Accept-Language** and **Content-Language** fields. A language tag is composed of 1 or more parts: A primary language tag and a possibly empty series of subtags:

```
language-tag  = primary-tag *( "-" subtag )
```

White space is not allowed within the tag and all tags are case-insensitive.

Example

Example tags include:

```
 en, en-US, en-cockney, i-cherokee, x-pig-latin
```

Where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

HTTP Messages

HTTP is based on client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages.

HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once connection is established, HTTP messages are passed in a format similar to that used by Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045]. These messages are consisted of requests from client to server and responses from server to client which will have following format:

```
HTTP-message   = <Request> | <Response> ; HTTP/1.1 messages
```

HTTP request and HTTP response use a generic message format of RFC 822 for transferring the required data. This generic message format consists of following four items:

- A Start-line
- Zero or more header fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

Following section will explain each of the entities used in HTTP message.

Message Start-Line

A start-line will have the following generic syntax:

```
start-line = Request-Line | Status-Line
```

We will discuss Request-Line and Status-Line while discussing HTTP Request and HTTP Response messages respectively. For now let's see the examples of start line in case of request and response:

```
GET /hello.htm HTTP/1.1     (This is Request-Line sent by the client)


HTTP/1.1 200 OK             (This is Status-Line sent by the server)
```

Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are following four types of HTTP message headers:

- **General-header**: These header fields have general applicability for both request and response messages.
- **Request-header**: These header fields are applicability only for request messages.
- **Response-header**: These header fields are applicability only for response messages.
- **Entity-header**: These header fields define metainformation about the entity-body or, if no body is present

All the above-mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (:) and the field value as follows:

```
message-header = field-name ":" [ field-value ]
```

Following are the examples of various header fields:

```
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain
```

Message Body

The message body part is optional for an HTTP message but if it is available then it is used to carry the entity-body associated with the request or response. If entity body is associated then usually Content-Type and Content-Length headers lines specify the nature of the body associated.

A message body is the one which carries actual HTTP request data (including form data and uploaded etc.) and HTTP response data from the server (including files, images, etc.). Following is a simple content of a message body:

```
<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>
```

HTTP Requests

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

- A Request line
- Zero or more header (General|Request|Entity) fields followed by CRLF
- An empty line (a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

Following section explains each of the entities used in HTTP message.

Message Request-Line

The Request-Line begins with a method token, followed by the Request-URI, the protocol version, and ending with CRLF. The elements are separated by space SP characters.

```
Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

Let us discuss each of the parts mentioned in Request-Line.

Request Method

The request **Method** indicates the method performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following methods are supported in HTTP/1.1:

S. No.	Method and Description
1	**GET** It is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** It is same as GET, but only transfers the status line and header section.
3	**POST** It is used to send data to the server. For example, customer information, file uploading, etc. using HTML forms.
4	**PUT** It replaces all current representations of the target resource with the uploaded content.
5	**DELETE** It removes all current representations of the target resource given by URI.
6	**CONNECT** It establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** It describes the communication options for the target resource.
8	**TRACE** It performs a message loop-back test along the path to the target resource.

Request-URI

The Request-URI is a Uniform Resource Identifier that identifies the resource upon which a request has to be applied. Following are the most commonly used forms to specify a URI:

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

S. No.	Method and Description
1	The asterisk ***** is used when HTTP request does not apply to a particular resource, but to the server itself. It is only allowed when the method does not necessarily apply to a resource. For example, **OPTIONS * HTTP/1.1**
2	The **absoluteURI** is used when HTTP request is being made to a proxy. The proxy is requested to forward the request or service it from a valid cache, and return the response. For example, **GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1**
3	The most common form of Request-URI is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve the resource above directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the lines: **GET /pub/WWW/TheProject.html HTTP/1.1** **Host: www.w3.org** **Note**: The absolute path cannot be empty. If none is present in the original URI, it must be given as "/" (the server root)

Request Header Fields

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers and the following important Request-header fields are available which can be used based on requirement:

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization

- Range
- Referer
- TE
- User-Agent

You can introduce your custom fields in case you are going to write your own custom Client and Web Server.

Request Message Examples

Now let us put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com:

```
GET /hello.htm HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plan HTML page from the server. Connection is a general-header and rest all headers are request headers. Following is another example where we send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Content-Type: application/x-www-form-urlencoded

Content-Length: length

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive


licenseID=string&content=string&/paramsXML=string
```

Here, the given URL */cgi-bin/process.cgi* is used to process the passed data and accordingly a response is retuned. The **content-type** tells the server that passed data is simple web form data and **length** is actual length of the data put in the message body. The following example shows how you can pass plan XML to your web server:

```
POST /cgi-bin/process.cgi HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com
```

```
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

HTTP Responses

After receiving and interpreting a request message, a server responds with an HTTP response message:

```
A Status-line

Zero or more header (General|Response|Entity) fields followed by CRLF

An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of
the header fields

Optionally a message-body
```

The following section explains each of the entities used in an HTTP message:

Message Status-Line

The Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Let us discuss each of the part mentioned in Status-Line.

HTTP Version

A server supporting HTTP version 1.1 returns the following version information:

```
HTTP-Version = HTTP/1.1
```

Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are five values for the first digit:

S. No.	Value and Description
1	**1xx: Informational** This means request received and continuing process.
2	**2xx: Success** This means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** This means further action must be taken in order to complete the request.
4	**4xx: Client Error** This means the request contains bad syntax or cannot be fulfilled
5	**5xx: Server Error** The server failed to fulfill an apparently valid request

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes.

Response Header Fields

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

- Accept-Ranges
- Age
- ETag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

You can introduce your custom fields in case you wish to write your own custom Web Client and Server.

Response Message Examples

Now let us put it all together to form an HTTP response for a request to fetch **hello.htm** page from the web server running on tutorialspoint.com

```
HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

Content-Length: 88

Content-Type: text/html

Connection: Closed


<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>
```

Following is an example of HTTP response message shows error condition when the web server could not find a requested page:

```
HTTP/1.1 404 Not Found

Date: Sun, 18 Oct 2012 10:36:20 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 230

Connection: Closed

Content-Type: text/html; charset=iso-8859-1


<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html>

<head>

   <title>404 Not Found</title>

</head>

<body>

   <h1>Not Found</h1>

   <p>The requested URL /t.html was not found on this server.</p>

</body>

</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in a given HTTP request:

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>400 Bad Request</title>
</head>
<body>
   <h1>Bad Request</h1>
   <p>Your browser sent a request that this server could not understand.<p>
   <p>The request line contained invalid characters following the protocol
string.<p>
</body>
</html>
```

HTTP Methods

The set of common methods for HTTP/1.1 is defined below and this set can be expanded based on requirement. These method names are case sensitive and they must be used in uppercase.

S. No.	Method and Description
1	**GET** It is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** It is same as GET, but only transfers the status line and header section.

3	**POST** It is used to send data to the server. For example, customer information, file uploading etc. using HTML forms.
4	**PUT** It replaces all current representations of the target resource with the uploaded content.
5	**DELETE** It removes all current representations of the target resource given by URI.
6	**CONNECT** It establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** It describes the communication options for the target resource.
8	**TRACE** It performs a message loop-back test along the path to the target resource.

GET Method

It retrieves data from a web server by specifying parameters in the URL portion of the request. This is the main method used for document retrieval. The following example makes use of GET method to fetch **hello.htm**:

```
GET /hello.htm HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive
```

The following server response is issued against the above GET request:

```
HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Vary: Authorization,Accept

Accept-Ranges: bytes

Content-Length: 88

Content-Type: text/html
```

```
Connection: Closed


<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HEAD Method

It is functionally similar to GET, except that the server replies with a response line and headers, but no entity-body. The following example makes use of HEAD method to fetch header information about **hello.htm**:

```
HEAD /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

The following server response is issued against the above GET request:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

You can notice that the server does not send any data after header.

POST Method

It is used when you want to send some data to the server. For example, file update, form data etc. The following simple example makes use of POST method to send a form data to the server which is processed by a **process.cgi** and finally a response is returned:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

Server side script **process.cgi** processes the passed data and sends the following response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Request Processed Successfully</h1>
</body>
</html>
```

PUT Method

The PUT method is used to request the server to store the included entity-body at a location specified by the given URL. The following example requests server to save the given entity-boy in **hello.htm** at the root of the server:

```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

The server stores the given entity-body in **hello.htm** file and sends the following response back to the client:

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```

DELETE Method

The DELETE method is used to request the server to delete file at a location specified by the given URL. The following example requests server to delete the given file **hello.htm** at the root of the server:

```
DELETE /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

The server deletes the mentioned file **hello.htm** and sends the following response back to the client:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

CONNECT Method

It is used by the client to establish a network connection to a web server over HTTP. The following example requests a connection with a web server running on host tutorialspoint.com:

```
CONNECT www.tutorialspoint.com HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The connection is established with the server and the following response is sent back to the client:

```
HTTP/1.1 200 Connection established
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
```

OPTIONS Method

It is used by the client to find out what are the HTTP methods and other options supported by a web server. The client can specify a URL for the OPTIONS method, or an asterisk (*) to refer to the entire server. The following example requests a list of methods supported by a web server running on tutorialspoint.com:

```
OPTIONS * HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server sends information based on the current configuration of the server, for example:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: httpd/unix-directory
```

TRACE Method

It is used to echo the contents of an HTTP Request back to the requester which can be used for debugging purpose at the time of development. The following example shows the usage of TRACE method:

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server will send the following message in response of the above request:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

HTTP Status Codes

The Status-Code element in a server response, is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are five values for the first digit:

S. No.	Code and Description
1	**1xx: Informational** It means the request was received and it is in process.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It implies further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes. The following list contains all the status codes:

1xx: Information

Message	Description
100 Continue	Only a part of the request is received by the server, but as long as it has not been rejected, the client should continue with the request.
101 Switching Protocols	The server switches protocol.

2xx: Successful

Message	Description
200 OK	The request is OK.
201 Created	The request is complete, and a new resource is created.
202 Accepted	The request is accepted for processing, but the processing is not complete.
203 Non-authoritative Information	The information in the entity header is from a local or third-party copy, not from the original server.
204 No Content	A status code and header are given in the response, but there is no entity-body in the reply.

205 Reset Content	The browser should clear the form used for this transaction for additional input.
206 Partial Content	The server is returning partial data of the size requested. It is used in response to a request specifying a*Range* header. The server must specify the range included in the response with the *Content-Range* header.

3xx: Redirection

Message	Description
300 Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses are available.
301 Moved Permanently	The requested page has moved to a new URL.
302 Found	The requested page has moved temporarily to a new URL.
303 See Other	The requested page can be found under a different URL.
304 Not Modified	This is the response code to an *If-Modified-Since* or *If-None-Match* header, where the URL has not been modified since the specified date.
305 Use Proxy	The requested URL must be accessed through the proxy mentioned in the *Location* header.
306 *Unused*	This code was used in a previous version. It is no longer used, but the code is reserved
307 Temporary Redirect	The requested page has moved temporarily to a new URL.

4xx: Client Error

Message	Description
400 Bad Request	The server did not understand the request.
401 Unauthorized	The requested page needs a username and a password.
402 Payment Required	*You cannot use this code yet.*

403 Forbidden	Access is forbidden to the requested page.
404 Not Found	The server cannot find the requested page.
405 Method Not Allowed	The method specified in the request is not allowed.
406 Not Acceptable	The server can only generate a response that is not accepted by the client.
407 Proxy Authentication Required	You must authenticate with a proxy server before this request can be served.
408 Request Timeout	The request took longer than the server was prepared to wait.
409 Conflict	The request could not be completed because of a conflict.
410 Gone	The requested page is no longer available.
411 Length Required	The "Content-Length" is not defined. The server will not accept the request without it.
412 Precondition Failed	The precondition given in the request evaluated is false by the server.
413 Request Entity Too Large	The server will not accept the request, because the request entity is too large.
414 Request-url Too Long	The server will not accept the request, because the URL is too long. It occurs when you convert a "post" request to a "get" request with a long query information.
415 Unsupported Media Type	The server will not accept the request, because the media type is not supported.
416 Requested Range Not Satisfiable	The requested byte range is not available and is out of bounds.
417 Expectation Failed	The expectation given in an Expect request-header field could not be met by this server.

5xx: Server Error

Message	Description
500 Internal Server Error	The request was not completed. The server met an unexpected condition.
501 Not Implemented	The request was not completed. The server did not support the functionality required.
502 Bad Gateway	The request was not completed. The server received an invalid response from the upstream server.
503 Service Unavailable	The request was not completed. The server is temporarily overloading or down.
504 Gateway Timeout	The gateway has timed out.
505 HTTP Version Not Supported	The server does not support the "http protocol" version.

HTTP Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- **General-header:** These header fields have general applicability for both request and response messages.
- **Client Request-header:** These header fields are applicability only for request messages.
- **Server Response-header:** These header fields are applicability only for response messages.
- **Entity-header:** These header fields define meta information about the entity-body or, if body is not present.

General Headers

Let us now go through the general headers in detail.

Cache-control

The Cache-Control general-header field is used to specify directives that must be obeyed by all caching system. The syntax is as follows:

```
Cache-Control : cache-request-directive|cache-response-directive
```

An HTTP clients or servers can use the **Cache-control** general header to specify parameters for the cache or to request certain kinds of documents from the cache. The caching directives are specified in a comma-separated list. For example:

```
Cache-control: no-cache
```

There are following important cache request directives which can be used by the client in its HTTP request:

S. No.	Cache Request Directive and Description
1	**no-cache** A cache must not use the response to satisfy a subsequent request without successful revalidation with the origin server.
2	**no-store** The cache should not store anything about the client request or server response.
3	**max-age = seconds** Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds.
4	**max-stale [= seconds]** Indicates that the client is willing to accept a response that has exceeded its expiration time. If seconds are given, it must not be expired by more than that time.
5	**min-fresh = seconds** Indicates that the client is willing to accept a response whose freshness lifetime is no less than its current age plus the specified time in seconds.
6	**no-transform** Do not convert the entity-body.
7	**only-if-cached** Do not retrieve new data. The cache can send a document only if it is in the cache, and should not contact the origin-server to see if a newer copy exists.

There are following important cache response directives which can be used by the server in its HTTP response:

S. No.	Cache Request Directive and Description
1	**public** Indicates that the response may be cached by any cache.
2	**private** Indicates that all or part of the response message is intended for a single user and must not be cached by a shared cache.

3	**no-cache** A cache must not use the response to satisfy a subsequent request without successful revalidation with the origin server.
4	**no-store** The cache should not store anything about the client request or server response.
5	**no-transform** Do not convert the entity-body.
6	**must-revalidate** The cache must verify the status of stale documents before using it and expired one should not be used.
7	**proxy-revalidate** The proxy-revalidate directive has the same meaning as the must-revalidate directive, except that it does not apply to non-shared user agent caches.
8	**max-age = seconds** Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds.
9	**s-maxage = seconds** The maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header. The s-maxage directive is always ignored by a private cache.

Connection

The Connection general-header field allows the sender to specify options that are desired for that particular connection and must not be communicated by proxies over further connections. Following is the simple syntax of using a connection header:

```
Connection : "Connection"
```

HTTP/1.1 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. For example:

```
Connection: close
```

By default, HTTP 1.1 uses persistent connections, where the connection does not automatically close after a transaction. HTTP 1.0, on the other hand, does not have persistent connections by default. If a 1.0 client wishes to use persistent connections, it uses the **keep-alive** parameter as follows:

```
Connection: keep-alive
```

Date

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036

Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

Here first format is the most preferred one.

Pragma

The Pragma general-header field is used to include implementation-specific directives that might apply to any recipient along the request/response chain. For example:

```
Pragma: no-cache
```

The only directive defined in HTTP/1.0 is the no-cache directive and is maintained in HTTP 1.1 for backward compatibility. No new Pragma directives will be defined in the future.

Trailer

The Trailer general field value indicates that the given set of header fields is present in the trailer of a message encoded with chunked transfer-coding. Following is the simple syntax of using a Trailer header field:

```
Trailer : field-name
```

Message header fields listed in the Trailer header field must not include the following header fields:

- Transfer-Encoding
- Content-Length
- Trailer

Transfer-Encoding

The *Transfer-Encoding* general-header field indicates what type of transformation has been applied to the message body in order to safely transfer it between the sender and the recipient. This is not the same as content-encoding because transfer-encodings are a property of the message, not of the entity-body. The following syntax shows using Transfer-Encoding header field:

```
Transfer-Encoding: chunked
```

All transfer-coding values are case-insensitive.

Upgrade

The *Upgrade* general-header allows the client to specify what additional communication protocols it supports and would like to use if the server finds it appropriate to switch protocols. For example:

```
Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

The Upgrade header field is intended to provide a simple mechanism for transition from HTTP/1.1 to some other, incompatible protocol

Via

The *Via* general-header must be used by gateways and proxies to indicate the intermediate protocols and recipients. For example, a request message could be sent from an HTTP/1.0 user agent to an internal proxy code-named "fred", which uses HTTP/1.1 to forward the request to a public proxy at nowhere.com, which completes the request by forwarding it to the origin server at www.ics.uci.edu. The request received by www.ics.uci.edu would then have the following Via header field:

```
Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
```

The Upgrade header field is intended to provide a simple mechanism for transition from HTTP/1.1 to some other, incompatible protocol

Warning

The *Warning* general-header is used to carry additional information about the status or transformation of a message which might not be reflected in the message. A response may carry more than one Warning header.

```
Warning : warn-code SP warn-agent SP warn-text SP warn-date
```

Client Request Headers

Accept

The *Accept* request-header field can be used to specify certain media types which are acceptable for the response. Following is the general syntax:

```
Accept: type/subtype [q=qvalue]
```

Multiple media types can be listed separated by commas and the optional qvalue represents an acceptable quality level for accept types on a scale of 0 to 1. Following is an example:

```
Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8, text/x-c
```

This would be interpreted as **text/html** and **text/x-c** are the preferred media types, but if they do not exist, then send the **text/x-dvi** entity, and if that does not exist, send the **text/plain** entity.

Accept-Charset

The *Accept-Charset* request-header field can be used to indicate what character sets are acceptable for the response. Following is the general syntax:

```
Accept-Charset: character_set [q=qvalue]
```

Multiple character sets can be listed separated by commas and the optional qvalue represents an acceptable quality level for nonpreferred character sets on a scale of 0 to 1. Following is an example:

```
Accept-Charset: iso-8859-5, unicode-1-1; q=0.8
```

The special value "*", if present in the **Accept-Charset** field, matches every character set and if no **Accept-Charset** header is present, the default is that any character set is acceptable.

Accept-Encoding

The *Accept-Encoding* request-header field is similar to Accept, but restricts the content-codings that are acceptable in the response. Following is the general syntax:

```
Accept-Encoding: encoding types
```

Following are examples:

```
Accept-Encoding: compress, gzip

Accept-Encoding:

Accept-Encoding: *

Accept-Encoding: compress;q=0.5, gzip;q=1.0

Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

Accept-Language

The *Accept-Language* request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request. Following is the general syntax:

```
Accept-Language: language [q=qvalue]
```

Multiple languages can be listed separated by commas and the optional qvalue represents an acceptable quality level for nonpreferred languages on a scale of 0 to 1. Following is an example:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

Authorization

The *Authorization* request-header field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested. Following is the general syntax:

```
Authorization : credentials
```

The HTTP/1.0 specification defines the BASIC authorization scheme, where the authorization parameter is the string of **username:password** encoded in base 64. Following is an example:

```
Authorization: BASIC Z3Vlc3Q6Z3Vlc3QxMjM=
```

The value decodes into is **guest:guest123** where **guest** is user ID and **guest123** is the password.

Cookie

The *Cookie* request-header field value contains a name/value pair of information stored for that URL. Following is the general syntax:

```
Cookie: name=value
```

Multiple cookies can be specified separated by semicolons as follows:

```
Cookie: name1=value1;name2=value2;name3=value3
```

Expect

The *Expect* request-header field is used to indicate that particular server behaviors are required by the client. Following is the general syntax:

```
Expect : 100-continue | expectation-extension
```

If a server receives a request containing an Expect field that includes an expectation-extension that it does not support, it must respond with a 417 (Expectation Failed) status.

From

The *From* request-header field contains an Internet e-mail address for the human user who controls the requesting user agent. Following is a simple example:

```
From: webmaster@w3.org
```

This header field may be used for logging purposes and as a means for identifying the source of invalid or unwanted requests.

Host

The *Host* request-header field is used to specify the Internet host and port number of the resource being requested. Following is the general syntax:

```
Host : "Host" ":" host [ ":" port ] ;
```

A **host** without any trailing port information implies the default port, which is 80. For example, a request on the origin server for *http://www.w3.org/pub/WWW/* would be:

```
GET /pub/WWW/ HTTP/1.1

Host: www.w3.org
```

If-Match

The *If-Match* request-header field is used with a method to make it conditional. This header request the server to perform the requested method only if given value in this tag matches the given entity tags represented by **ETag**. Following is the general syntax:

```
If-Match : entity-tag
```

An asterisk (*) matches any entity, and the transaction continues only if the entity exists. Following are possible examples:

```
If-Match: "xyzzy"

If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"

If-Match: *
```

If none of the entity tags match, or if "*" is given and no current entity exists, the server must not perform the requested method, and must return a 412 (Precondition Failed) response.

If-Modified-Since

The *If-Modified-Since* request-header field is used with a method to make it conditional. If the requested URL has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body. Following is the general syntax:

```
If-Modified-Since : HTTP-date
```

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If none of the entity tags match, or if "*" is given and no current entity exists, the server must not perform the requested method, and must return a 412 (Precondition Failed) response.

If-None-Match

The *If-None-Match* request-header field is used with a method to make it conditional. This header request the server to perform the requested method only if one of the given value in this tag matches the given entity tags represented by **ETag**. Following is the general syntax:

```
If-None-Match : entity-tag
```

An asterisk (*) matches any entity, and the transaction continues only if the entity does not exist. Following are possible examples:

```
If-None-Match: "xyzzy"

If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"

If-None-Match: *
```

If-Range

The *If-Range* request-header field can be used with a conditional GET to request only the portion of the entity that is missing, if it has not been changed, and the entire entity if it has changed. Following is the general syntax:

```
If-Range : entity-tag | HTTP-date
```

Either an entity tag or a date can be used to identify the partial entity already received. For example:

```
If-Range: Sat, 29 Oct 1994 19:43:31 GMT
```

Here if the document has not been modified since the given date, the server returns the byte range given by the Range header otherwise, it returns all of the new document.

If-Unmodified-Since

The *If-Unmodified-Since* request-header field is used with a method to make it conditional. Following is the general syntax:

```
If-Unmodified-Since : HTTP-date
```

If the requested resource has not been modified since the time specified in this field, the server should perform the requested operation as if the If-Unmodified-Since header were not present. For example:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If the request normally would result in anything other than a 2xx or 412 status, the *If-Unmodified-Since* header should be ignored.

Max-Forwards

The *Max-Forwards* request-header field provides a mechanism with the TRACE and OPTIONS methods to limit the number of proxies or gateways that can forward the request to the next inbound server. Following is the general syntax:

```
Max-Forwards : n
```

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message may be forwarded. This is useful for debugging with the TRACE method, avoiding infinite loops. For example:

```
Max-Forwards : 5
```

The Max-Forwards header field may be ignored for all other methods defined in HTTP specification.

Proxy-Authorization

The *Proxy-Authorization* request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. Following is the general syntax:

```
Proxy-Authorization : credentials
```

The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

Range

The *Range* request-header field specifies the partial range(s) of the content requested from the document. Following is the general syntax:

```
Range: bytes-unit=first-byte-pos "-" [last-byte-pos]
```

The first-byte-pos value in a byte-range-spec gives the byte-offset of the first byte in a range. The last-byte-pos value gives the byte-offset of the last byte in the range; that is, the byte positions specified are inclusive. You can specify a byte-unit as bytes Byte offsets start at zero. Following are a simple examples:

```
- The first 500 bytes
Range: bytes=0-499

- The second 500 bytes
Range: bytes=500-999

- The final 500 bytes
Range: bytes=-500

- The first and last bytes only
Range: bytes=0-0,-1
```

Multiple ranges can be listed, separated by commas. If the first digit in the comma-separated byte range(s) is missing, the range is assumed to count from the end of the document. If the second digit is missing, the range is byte n to the end of the document.

Referer

The *Referer* request-header field allows the client to specify the address (URI) of the resource from which the URL has been requested. Following is the general syntax:

```
Referer : absoluteURI | relativeURI
```

Following is a simple example:

```
Referer: http://www.tutorialspoint.org/http/index.htm
```

If the field value is a relative URI, it should be interpreted relative to the *Request-URI*.

TE

The *TE* request-header field indicates what extension *transfer-coding* it is willing to accept in the response and whether or not it is willing to accept trailer fields in a chunked *transfer-coding*. Following is the general syntax:

```
TE   : t-codings
```

The presence of the keyword "trailers" indicates that the client is willing to accept trailer fields in a chunked transfer-coding and it is specified either of the ways:

```
TE: deflate

TE:

TE: trailers, deflate;q=0.5
```

If the TE field-value is empty or if no TE field is present, the only transfer-coding is *chunked*. A message with no transfer-coding is always acceptable.

User-Agent

The *User-Agent* request-header field contains information about the user agent originating the request. Following is the general syntax:

```
User-Agent : product | comment
```

Example:

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Server Response Headers

They are a given:

Accept-Ranges

The *Accept-Ranges* response-header field allows the server to indicate its acceptance of range requests for a resource. Following is the general syntax:

```
Accept-Ranges  : range-unit | none
```

For example, a server that accept byte-range requests may send

```
Accept-Ranges: bytes
```

Servers that do not accept any kind of range request for a resource may send:

```
Accept-Ranges: none
```

This will advise the client not to attempt a range request.

Age

The *Age* response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. Following is the general syntax:

```
Age : delta-seconds
```

Age values are non-negative decimal integers, representing time in seconds. Following is a simple example:

```
Age: 1030
```

An HTTP/1.1 server that includes a cache must include an Age header field in every response generated from its own cache.

ETag

The *ETag* response-header field provides the current value of the entity tag for the requested variant. Following is the general syntax:

```
ETag :  entity-tag
```

Following are simple examples:

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

Location

The *Location* response-header field is used to redirect the recipient to a location other than the Request-URI for completion. Following is the general syntax:

```
Location : absoluteURI
```

Following is a simple example:

```
Location: http://www.tutorialspoint.org/http/index.htm
```

The Content-Location header field differs from Location in that the Content-Location identifies the original location of the entity enclosed in the request.

Proxy-Authenticate

The *Proxy-Authenticate* response-header field must be included as part of a 407 (Proxy Authentication Required) response. Following is the general syntax:

```
Proxy-Authenticate  : challenge
```

Retry-After

The *Retry-After* response-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. Following is the general syntax:

```
Retry-After : HTTP-date | delta-seconds
```

Following are two simple examples:

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT

Retry-After: 120
```

In the latter example, the delay is 2 minutes.

Server

The *Server* response-header field contains information about the software used by the origin server to handle the request. Following is the general syntax:

```
Server : product | comment
```

Following is a simple example:

```
Server: Apache/2.2.14 (Win32)
```

If the response is being forwarded through a proxy, the proxy application must not modify the Server response-header.

Set-Cookie

The *Set-Cookie* response-header field contains a name/value pair of information to retain for this URL. Following is the general syntax:

```
Set-Cookie: NAME=VALUE; OPTIONS
```

Set-Cookie response header comprises the token Set-Cookie:, followed by a comma-separated list of one or more cookies. Here are possible values you can specify as options:

S. No.	Options and Description
1	Comment=comment This option can be used to specify any comment associated with the cookie.
2	Domain=domain The Domain attribute specifies the domain for which the cookie is valid.
3	Expires=Date-time The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser

4	Path=path The Path attribute specifies the subset of URLs to which this cookie applies.
5	Secure This instructs the user agent to return the cookie only under a secure connection.

Following is an example of a simple cookie header generated by the server:

```
Set-Cookie: name1=value1,name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

Vary

The *Vary* response-header field specifies that the entity has multiple sources and may therefore vary according to specified list of request header(s). Following is the general syntax:

```
Vary : field-name
```

You can specify multiple headers separated by commas and a value of asterisk "*" signals that unspecified parameters not limited to the request-headers. Following is a simple example:

```
Vary: Accept-Language, Accept-Encoding
```

Here, the field names are case-insensitive.

WWW-Authenticate

The *WWW-Authenticate* response-header field must be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI. Following is the general syntax:

```
WWW-Authenticate : challenge
```

WWW- Authenticate field value as it might contain more than one challenge, or if more than one WWW-Authenticate header field is provided, the contents of a challenge itself can contain a comma-separated list of authentication parameters. Following is a simple example:

```
WWW-Authenticate: BASIC realm="Admin"
```

Entity Headers

Allow

The *Allow* entity-header field lists the set of methods supported by the resource identified by the Request-URI. Following is the general syntax:

```
Allow : Method
```

You can specify multiple method separated by commas. Following is a simple example:

```
Allow: GET, HEAD, PUT
```

This field cannot prevent a client from trying other methods.

Content-Encoding

The *Content-Encoding* entity-header field is used as a modifier to the media-type. Following is the general syntax:

```
Content-Encoding : content-coding
```

The content-coding is a characteristic of the entity identified by the Request-URI. Following is a simple example:

```
Content-Encoding: gzip
```

If the content-coding of an entity in a request message is not acceptable to the origin server, the server should respond with a status code of 415 (Unsupported Media Type).

Content-Language

The *Content-Language* entity-header field describes the natural language(s) of the intended audience for the enclosed entity. Following is the general syntax:

```
Content-Language : language-tag
```

Multiple languages may be listed for content that is intended for multiple audiences. Following is a simple example:

```
Content-Language: mi, en
```

The primary purpose of Content-Language is to allow a user to identify and differentiate entities according to the user's own preferred language.

Content-Length

The *Content-Length* entity-header field indicates the size of the entity-body, in decimal number of OCTETs, sent to the recipient or, in the case of the HEAD method, the size of the entity-body that would have been sent had the request been a GET. Following is the general syntax:

```
Content-Length : DIGITS
```

Following is a simple example:

```
Content-Length: 3495
```

Any Content-Length greater than or equal to zero is a valid value.

Content-Location

The *Content-Location* entity-header field may be used to supply the resource location for the entity enclosed in the message when that entity is accessible from a location separate from the requested resource's URI. Following is the general syntax:

```
Content-Location:  absoluteURI | relativeURI
```

Following is a simple example:

```
Content-Location: http://www.tutorialspoint.org/http/index.htm
```

The value of Content-Location also defines the base URI for the entity.

Content-MD5

The *Content-MD5* entity-header field may be used to supply an MD5 digest of the entity, for checking the integrity of the message upon receipt. Following is the general syntax:

```
Content-MD5  : md5-digest using base64 of 128 bit MD5 digest as per RFC 1864
```

Following is a simple example:

```
Content-MD5  : 8c2d46911f3f5a326455f0ed7a8ed3b3
```

The MD5 digest is computed based on the content of the entity-body, including any content-coding that has been applied, but not including any transfer-encoding applied to the message-body.

Content-Range

The *Content-Range* entity-header field is sent with a partial entity-body to specify where in the full entity-body the partial body should be applied. Following is the general syntax:

```
Content-Range : bytes-unit SP first-byte-pos "-" last-byte-pos
```

Examples of byte-content-range-spec values, assuming that the entity contains a total of 1234 bytes:

```
- The first 500 bytes:
Content-Range : bytes 0-499/1234


- The second 500 bytes:
Content-Range : bytes 500-999/1234


- All except for the first 500 bytes:
Content-Range : bytes 500-1233/1234


- The last 500 bytes:
Content-Range : bytes 734-1233/1234
```

When an HTTP message includes the content of a single range, this content is transmitted with a Content-Range header, and a Content-Length header showing the number of bytes actually transferred. For example,

```
HTTP/1.1 206 Partial content

Date: Wed, 15 Nov 1995 06:25:24 GMT

Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT

Content-Range: bytes 21010-47021/47022

Content-Length: 26012

Content-Type: image/gif
```

Content-Type

The *Content-Type* entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET. Following is the general syntax:

```
Content-Type : media-type
```

Following is an example:

```
Content-Type: text/html; charset=ISO-8859-4
```

Expires

The *Expires* entity-header field gives the date/time after which the response is considered stale. Following is the general syntax:

```
Expires : HTTP-date
```

Following is an example:

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

Last-Modified

The *Last-Modified* entity-header field indicates the date and time at which the origin server believes the variant was last modified. Following is the general syntax:

```
Last-Modified: HTTP-date
```

Following is an example:

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

HTTP Security

HTTP is used for a communication over the internet, so application developers, information providers, and the users should be aware of the security limitations in HTTP/1.1. This discussion does not include definitive solutions to the problems mentioned here but it does make some suggestions for reducing security risks.

Leakage of Personal Information

HTTP clients are often privy to large amounts of personal information such as the user's name, location, mail address, passwords, encryption keys etc. Hence as a user you need to be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

- All the confidential information should be stored at server side in encrypted form.
- Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes.
- Proxies which serve as a portal through a network firewall should take special precautions regarding the transfer of header information that identifies the hosts behind the firewall.
- The information sent in the Form field might conflict with the user's privacy interests or their site's security policy. Hence it should not be transmitted without the user being able to disable, enable, and modify the contents of the field.
- Clients should not include a *Referer* header field in a non-secure HTTP request if the referring page was transferred with a secure protocol.
- Authors of services which use the HTTP protocol should not use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI.

File and Path Names Based Attack

The document should be restricted to the documents returned by HTTP requests to be only those that were intended by the server administrators.

For example, UNIX, Microsoft Windows, and other operating systems use **..** as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the Request-URI if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server.

DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. The clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they must observe the TTL information reported by DNS. If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes.

Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content- Location headers in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

Authentication Credentials

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP/1.1 does not provide a method for a server to direct clients to discard these cached credentials, which is a big security risk.

It is recommended to make the use of password protection in screen savers, idle time-outs, and other methods which mitigate the security problems inherent in this problem.

Proxies and Caching

HTTP proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers.

Proxy operators should protect the systems on which proxies run as they would protect any system that contains or transports sensitive information.

Caching proxies provide additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Therefore, cache contents should be protected as sensitive information.

5. HTTPS Protocol Basics

HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) or HTTP over SSL is a web protocol developed by Netscape. It is not a protocol but it is just the result of layering the HTTP on top of SSL/TLS (Secure Socket Layer/Transport Layer Security).

In short, HTTPS = HTTP + SSL

When is HTTPS Required?

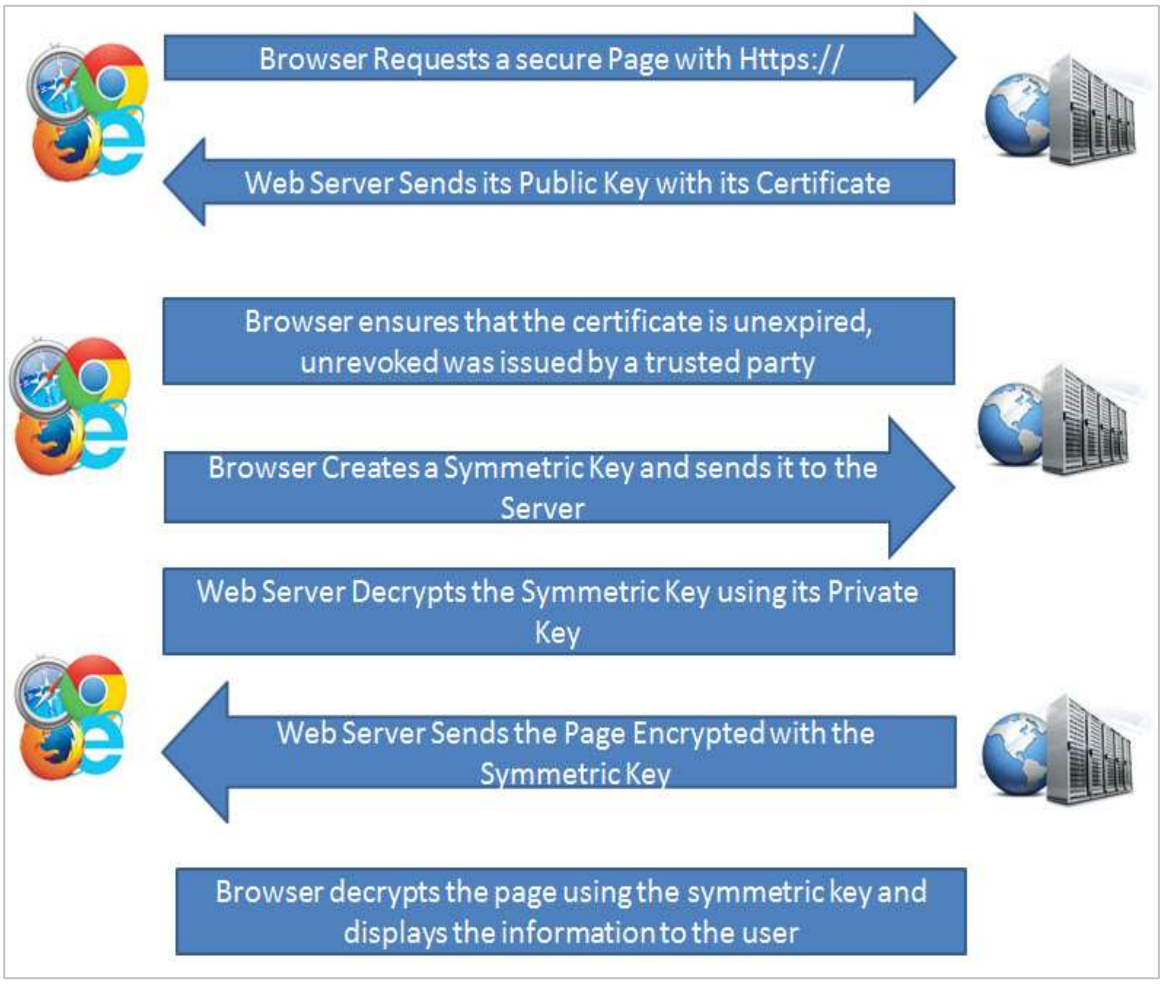
When we browse, we normally send and receive information using HTTP protocol. So this leads anyone to eavesdrop on the conversation between our computer and the web server. Many a times we need to exchange sensitive information which needs to be secured and to prevent unauthorized access.

Https protocol used in the following scenarios

- Banking Websites
- Payment Gateway
- Shopping Websites
- All Login Pages
- Email Apps

Basic Working of HTTPS

- Public key and signed certificates are required for the server in HTTPS Protocol.
- Client requests for the https:// page
- When using an https connection, the server responds to the initial connection by offering a list of encryption methods the webserver supports.
- In response, the client selects a connection method, and the client and server exchange certificates to authenticate their identities.
- After this is done, both webserver and client exchange the encrypted information after ensuring that both are using the same key, and the connection is closed.
- For hosting https connections, a server must have a public key certificate, which embeds key information with a verification of the key owner's identity.
- Almost all certificates are verified by a third party so that clients are assured that the key is always secure.


Browser Requests a secure Page with Https://
Web Server Sends its Public Key with its Certificate
Browser ensures that the certificate is unexpired, unrevoked was issued by a trusted party
Browser Creates a Symmetric Key and sends it to the Server
Web Server Decrypts the Symmetric Key using its Private Key
Web Server Sends the Page Encrypted with the Symmetric Key
Browser decrypts the page using the symmetric key and displays the information to the user

6. Encoding and Decoding

What is Encoding and Decoding?

Encoding is the process of putting a sequence of characters such as letters, numbers and other special characters into a specialized format for efficient transmission.

Decoding is the process of converting an encoded format back into the original sequence of characters. It is completely different from Encryption which we usually misinterpret.

Encoding and decoding are used in data communications and storage. Encoding should NOT be used for transporting sensitive information.

URL Encoding

URLs can only be sent over the Internet using the ASCII character-set and there are instances when URL contains special characters apart from ASCII characters, it needs to be encoded. URLs do not contain spaces and are replaced with a plus (+) sign or with %20.

ASCII Encoding

The Browser (client side) will encode the input according to the character-set used in the web-page and the default character-set in HTML5 is UTF-8.

Following table shows ASCII symbol of the character and its equal Symbol and finally its replacement which can be used in URL before passing it to the server:

ASCII	Symbol	Replacement
< 32		Encode with %xx where xx is the hexadecimal representation of the character.
32	space	+ or %20
33	!	%21
34	"	%22
35	#	%23
36	$	%24
37	%	%25
38	&	%26
39	'	%27

40	(	%28
41	)	%29
42	*	*
43	+	%2B
44	,	%2C
45	-	-
46	.	.
47	/	%2F
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	%3A
59	;	%3B
60	<	%3C
61	=	%3D
62	>	%3E

63	?	%3F
64	@	%40
65	A	A
66	B	B
67	C	C
68	D	D
69	E	E
70	F	F
71	G	G
72	H	H
73	I	I
74	J	J
75	K	K
76	L	L
77	M	M
78	N	N
79	O	O
80	P	P
81	Q	Q
82	R	R
83	S	S
84	T	T
85	U	U

86	V	V
87	W	W
88	X	X
89	Y	Y
90	Z	Z
91	[	%5B
92	\	%5C
93	]	%5D
94	^	%5E
95	_	_
96	`	%60
97	a	a
98	b	b
99	c	c
100	d	d
101	e	e
102	f	f
103	g	g
104	h	h
105	i	i
106	j	j
107	k	k
108	l	l

109	m	m
110	n	n
111	o	o
112	p	p
113	q	q
114	r	r
115	s	s
116	t	t
117	u	u
118	v	v
119	w	w
120	x	x
121	y	y
122	z	z
123	{	%7B
124	|	%7C
125	}	%7D
126	~	%7E
127		%7F
> 127		Encode with %xx where xx is the hexadecimal representation of the character

7. Cryptography

What is Cryptography?

Cryptography is the science to encrypt and decrypt data that enables the users to store sensitive information or transmit it across insecure networks so that it can be read only by the intended recipient.

Data which can be read and understood without any special measures is called **plaintext**, while the method of disguising plaintext in order to hide its substance is called **encryption**.

Encrypted plaintext is known as cipher text and process of reverting the encrypted data back to plain text is known as **decryption**.

- The science of analyzing and breaking secure communication is known as cryptanalysis. The people who perform the same also known as attackers.
- Cryptography can be either strong or weak and the strength is measured by the time and resources it would require to recover the actual plaintext.
- Hence an appropriate decoding tool is required to decipher the strong encrypted messages.
- There are some cryptographic techniques available with which even a billion computers doing a billion checks a second, it is not possible to decipher the text.
- As the computing power is increasing day by day, one has to make the encryption algorithms very strong in order to protect data and critical information from the attackers.

How Encryption Works?

A cryptographic algorithm works in combination with a key (can be a word, number, or phrase) to encrypt the plaintext and the same plaintext encrypts to different cipher text with different keys.

Hence, the encrypted data is completely dependent couple of parameters such as the strength of the cryptographic algorithm and the secrecy of the key.

Cryptography Techniques

Symmetric Encryption – Conventional cryptography, also known as conventional encryption, is the technique in which only one key is used for both encryption and decryption. For example, DES, Triple DES algorithms, MARS by IBM, RC2, RC4, RC5, RC6.

Asymmetric Encryption – It is Public key cryptography that uses a pair of keys for encryption: a public key to encrypt data and a private key for decryption. Public key is published to the people while keeping the private key secret. For example, RSA, Digital Signature Algorithm (DSA), Elgamal.

Hashing – Hashing is ONE-WAY encryption, which creates a scrambled output that cannot be reversed or at least cannot be reversed easily. For example, MD5 algorithm. It is used to create Digital Certificates, Digital signatures, Storage of passwords, Verification of communications, etc.

8. Same Origin Policy

Same Origin Policy (SOP) is an important concept in the web application security model.

What is Same Origin Policy?

As per this policy, it permits scripts running on pages originating from the same site which can be a combination of the following:

- Domain
- Protocol
- Port

Example

The reason behind this behavior is security. If you have *try.com* in one window and *gmail.com* in another window, then you DO NOT want a script from try.com to access or modify the contents of gmail.com or run actions in context of gmail on your behalf.

Below are webpages from the same origin. As explained before, the same origin takes domain/protocol/port into consideration.

- http://website.com
- http://website.com/
- http://website.com/my/contact.html

Below are webpages from a different origin.

- http://www.site.co.uk(another domain)
- http://site.org (another domain)
- https://site.com (another protocol)
- http://site.com:8080 (another port)

Same Origin policy Exceptions for IE

Internet Explorer has two major exceptions to SOP.

- The first one is related to 'Trusted Zones'. If both domains are in highly trusted zone then the Same Origin policy is not applicable completely.

- The second exception in IE is related to port. IE does not include port into Same Origin policy, hence the http://website.com and http://wesite.com:4444 are considered from the same origin and no restrictions are applied.

9. Testing Cookies

What is a Cookie?

A cookie is a small piece of information sent by a web server to store on a web browser so that it can later be read by the browser. This way, the browser remembers some specific personal information. If a Hacker gets hold of the cookie information, it can lead to security issues.

Properties of Cookies

Here are some important properties of cookies:

- They are usually small text files, given ID tags that are stored on your computer's browser directory.
- They are used by web developers to help users navigate their websites efficiently and perform certain functions.
- When the user browses the same website again, the data stored in the cookie is sent back to the web server to notify the website of the user's previous activities.
- Cookies are unavoidable for websites that have huge databases, need logins, have customizable themes.

Cookie Contents

The cookie contains the following information:

- The name of the server the cookie was sent from.
- The lifetime of the cookie.
- A value - usually a randomly generated unique number.

Types of Cookies

- Session **Cookies** - These cookies are temporary which are erased when the user closes the browser. Even if the user logs in again, a new cookie for that session is created.
- **Persistent cookies** - These cookies remain on the hard disk drive unless user wipes them off or they expire. The Cookie's expiry is dependent on how long they can last.

Testing Cookies

Here are the ways to test the cookies:

- **Disabling Cookies:** As a tester, we need to verify the access of the website after disabling cookies and to check if the pages are working properly. Navigating to all the pages of the website and watch for app crashes. It is also required to inform the user that cookies are required to use the site.

- **Corrupting Cookies:** Another testing to be performed is by corrupting the cookies. In order to do the same, one has to find the location of the site's cookie and manually edit it with fake / invalid data which can be used access internal information from the domain which in turn can then be used to hack the site.

- **Removing Cookies:** Remove all the cookies for the website and check how the website reacts to it.

- **Cross-Browser Compatibility:** It is also important to check that cookies are being written properly on all supported browsers from any page that writes cookies.

- **Editing Cookies:** If the application uses cookies to store login information then as a tester we should try changing the user in the cookie or address bar to another valid user. Editing the cookie should not let you log in to a different users account.

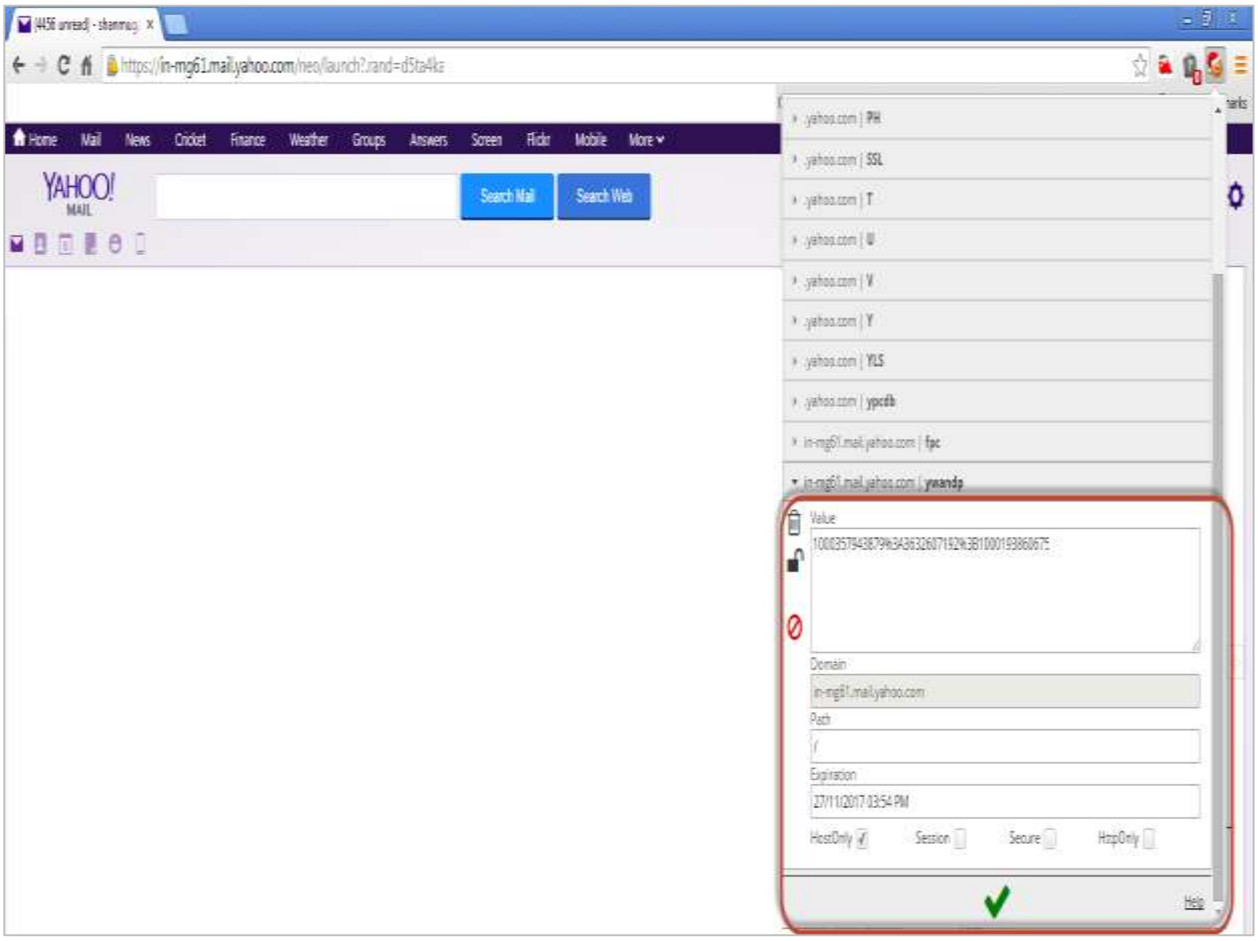
Viewing and Editing Cookies

Modern browsers support viewing/editing of the cookies inform within the Browser itself. There are plugins for mozilla/chrome using which we are able to perform the edit successfully.

- Edit cookies plugin for Firefox
- Edit This cookie plugin for chrome

The steps should be performed to Edit a cookie:

- Download the plugin for Chrome.
- Edit the cookie value just by accessing the 'edit this cookie' plugin from chrome as shown below.



.yahoo.com | PH

.yahoo.com | SSL

.yahoo.com | T

.yahoo.com | U

.yahoo.com | V

.yahoo.com | Y

.yahoo.com | YLS

.yahoo.com | ypcdb

in-mg61.mail.yahoo.com | fpc

in-mg61.mail.yahoo.com | ywandp

Value
1000357943879%3A3632607192%3B1000193860675

Domain
in-mg61.mail.yahoo.com

Path
/

Expiration
27/11/2017 03:54 PM

HostOnly ✓ Session ☐ Secure ☐ HttpOnly ☐

10. Hacking Web Applications

There are various methodologies/approaches which we can make use of as a reference for performing an attack.

Web Application - PenTesting Methodologies

One can take into account the following standards while developing an attack model.

Among the following list, OWASP is the most active and there are a number of contributors. We will focus on OWASP Techniques which each development team takes into consideration before designing a web app.

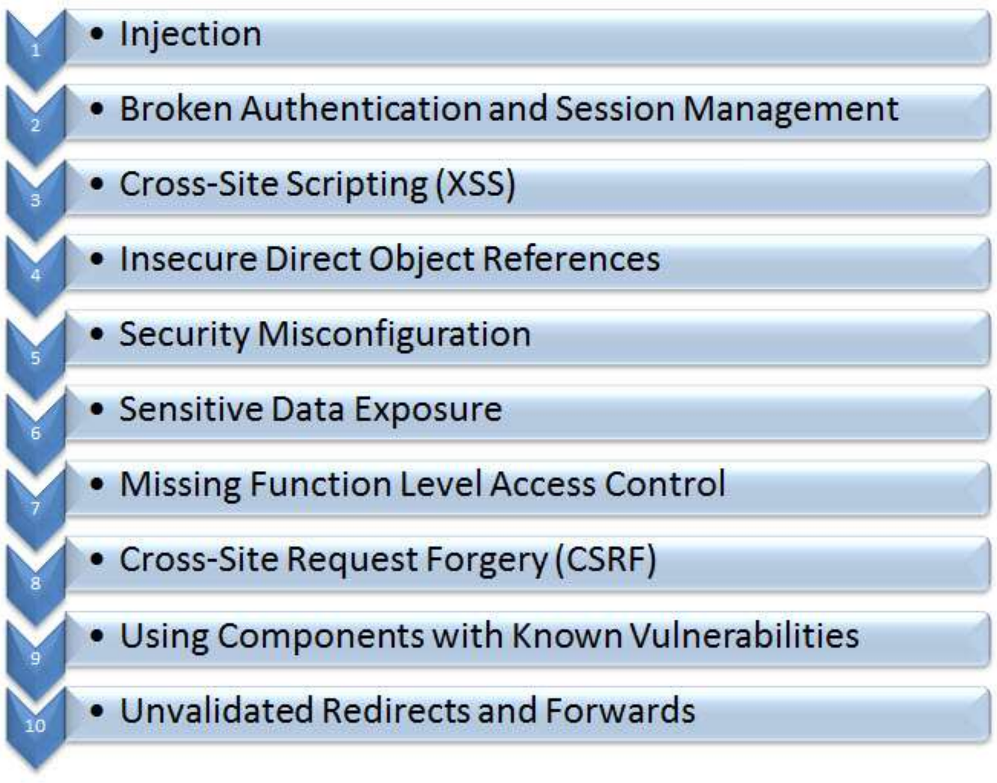
PTES - Penetration Testing Execution Standard

OSSTMM - Open Source Security Testing Methodology Manual

OWASP Testing Techniques - Open Web Application Security Protocol

OWASP Top 10

The Open Web Application Security Protocol team released the top 10 vulnerabilities that are more prevalent in web in the recent years. Below is the list of security flaws that are more prevalent in a web based application.



1
• Injection
2
• Broken Authentication and Session Management
3
• Cross-Site Scripting (XSS)
4
• Insecure Direct Object References
5
• Security Misconfiguration
6
• Sensitive Data Exposure
7
• Missing Function Level Access Control
8
• Cross-Site Request Forgery (CSRF)
9
• Using Components with Known Vulnerabilities
10
• Unvalidated Redirects and Forwards

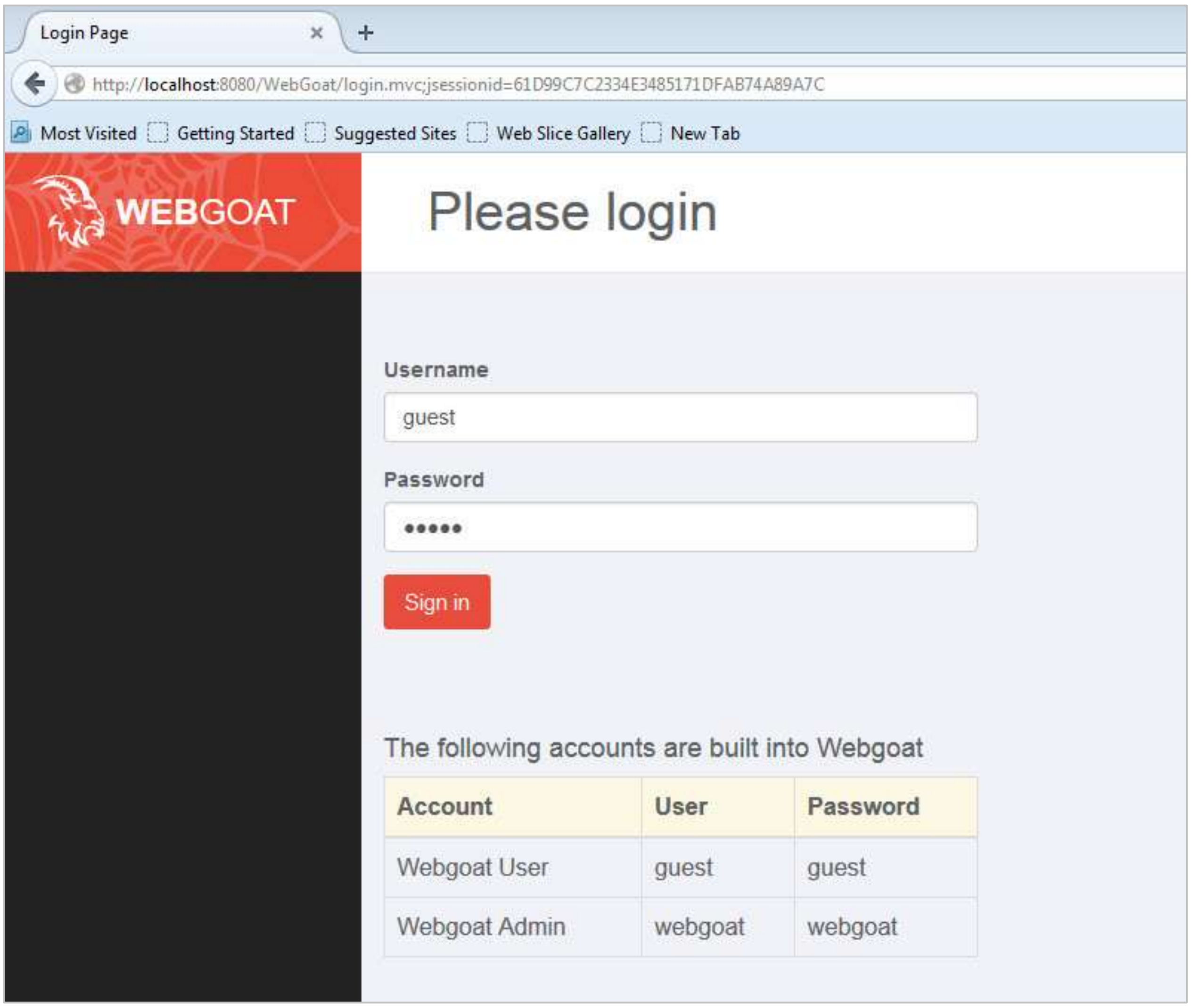
Application - Hands On

In order to understand each one of the techniques, let us work with a sample application. We will perform the attack on 'WebGoat', the J2EE application which is developed explicitly with security flaws for learning purposes.

The complete details about the webgoat project can be located at https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project.To Download the WebGoat Application, Navigate to https://github.com/WebGoat/WebGoat/wiki/Installation-(WebGoat-6.0) and go to downloads section.

To install the downloaded application, first ensure that you do not have any application running on Port 8080. It can be installed just using a single command - java -jar WebGoat-6.0.1-war-exec.jar. For more details, visit WebGoat Installation

Post Installation, you should be able to access the application by navigating to http://localhost:8080/WebGoat/attack and the page would be displayed as shown below.



The following accounts are built into Webgoat

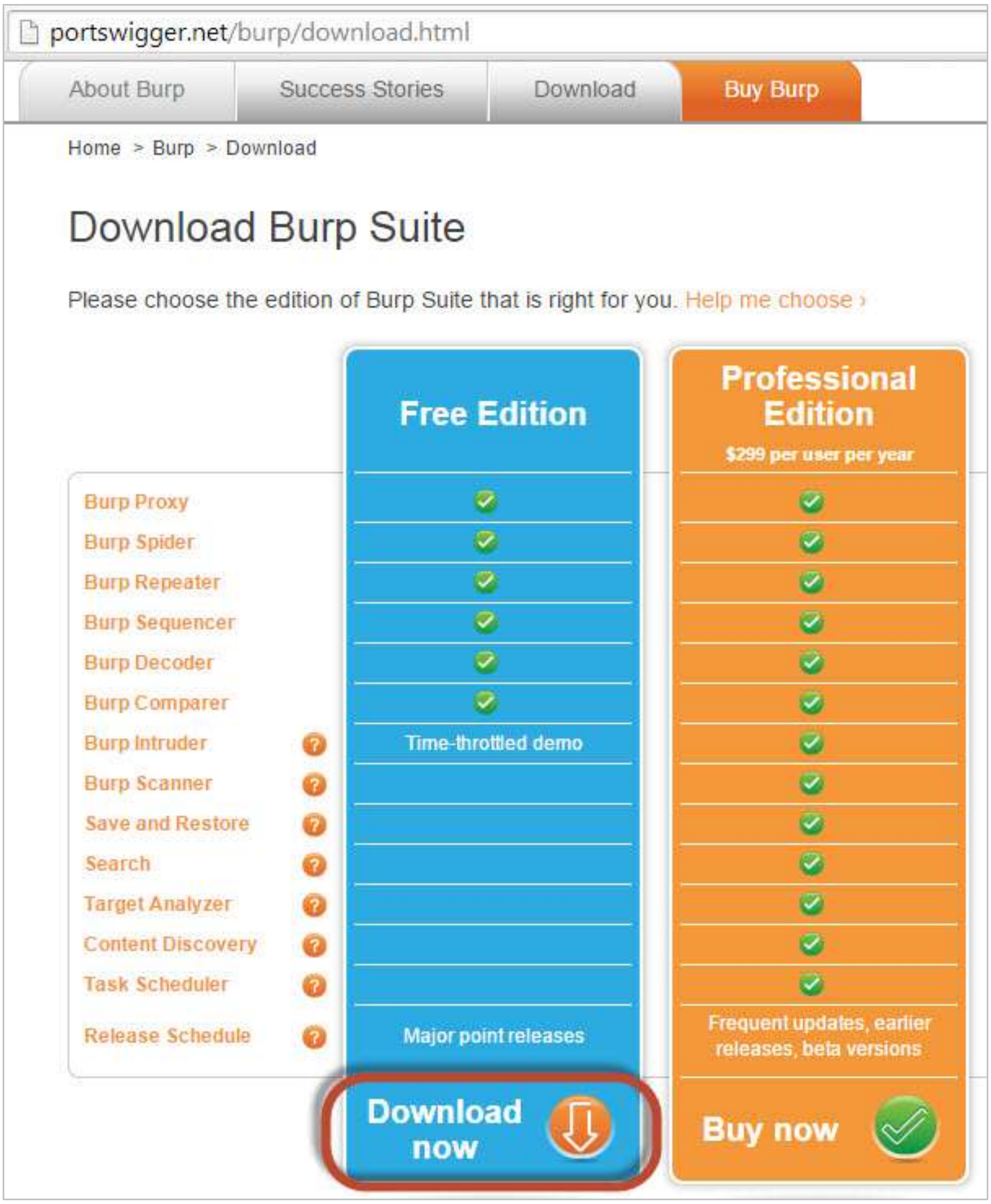
Account	User	Password
Webgoat User	guest	guest
Webgoat Admin	webgoat	webgoat

We can use the credentials of guest or admin as displayed in the login page.

Web Proxy

In order to intercept the traffic between client (Browser) and Server (System where Webgoat Application is hosted in our case), we need to use a web proxy. We will use Burp Proxy that can be downloaded from http://portswigger.net/burp/download.html

It is sufficient if you download the free version of burp suite as shown below.



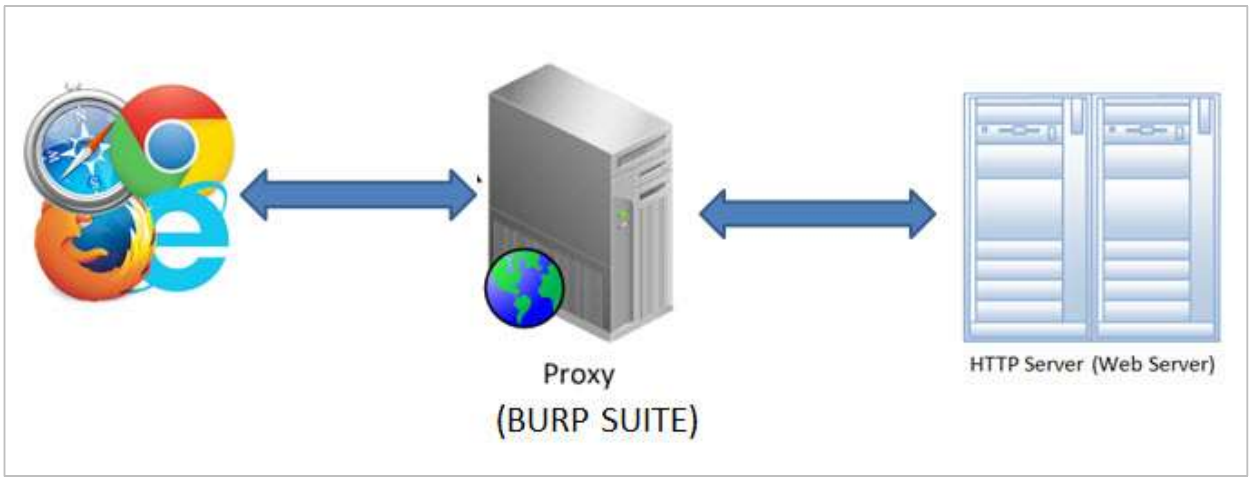
Download Burp Suite

Please choose the edition of Burp Suite that is right for you. Help me choose ›

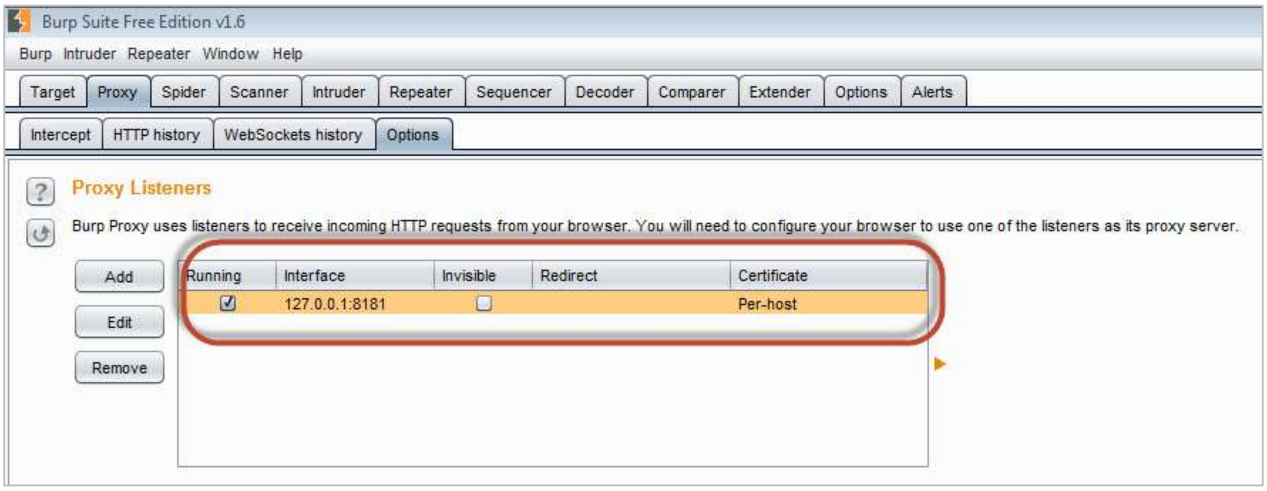
	Free Edition	Professional Edition $299 per user per year
Burp Proxy	✓	✓
Burp Spider	✓	✓
Burp Repeater	✓	✓
Burp Sequencer	✓	✓
Burp Decoder	✓	✓
Burp Comparer	✓	✓
Burp Intruder	Time-throttled demo	✓
Burp Scanner		✓
Save and Restore		✓
Search		✓
Target Analyzer		✓
Content Discovery		✓
Task Scheduler		✓
Release Schedule	Major point releases	Frequent updates, earlier releases, beta versions

Configuring Burp Suite

Burp Suite is a web proxy which can intercept each packet of information sent and received by the browser and webserver. This helps us to modify the contents before the client sends the information to the Web-Server.



1. The App is installed on port 8080 and Burp is installed on port 8181 as shown below. Launch Burp suite and make the following settings in order to bring it up in port 8181 as shown below.

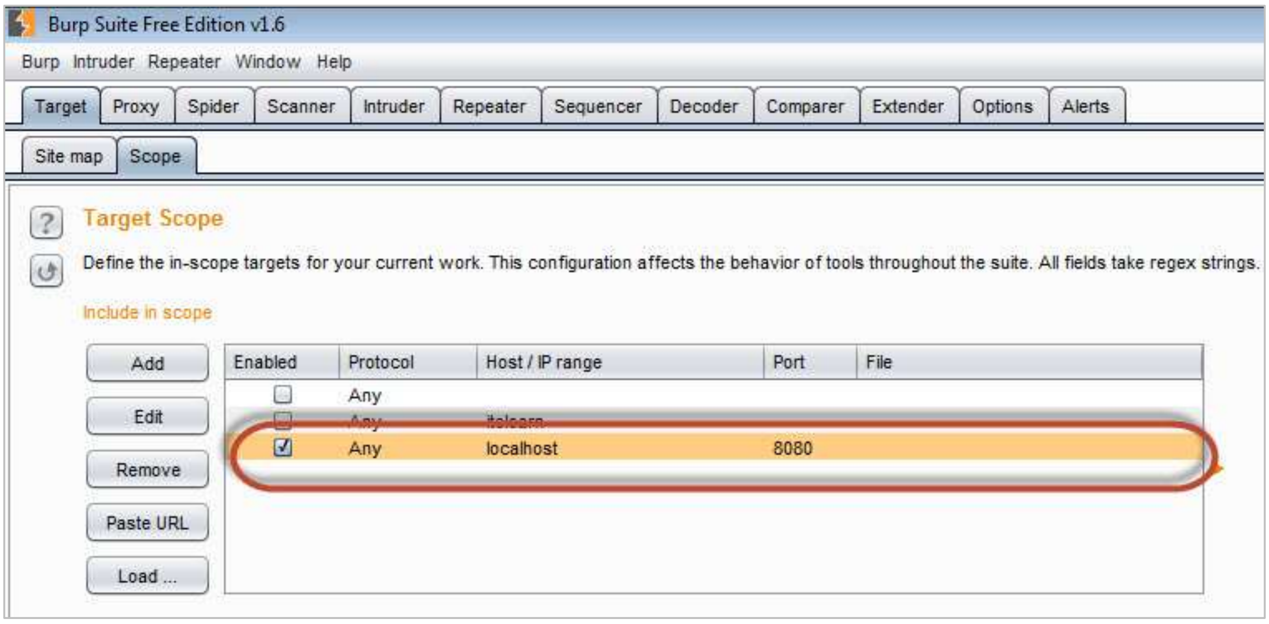

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Add
Edit
Remove

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Running	Interface	Invisible	Redirect	Certificate
✓	127.0.0.1:8181	☐		Per-host

2. We should ensure that the Burp is listening to Port#8080 where the application is installed so that Burp suite can intercept the traffic. This settings should be done on the scope tab of the Burp Suite as shown below.


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Site map Scope
Add
Edit
Remove
Paste URL
Load ...

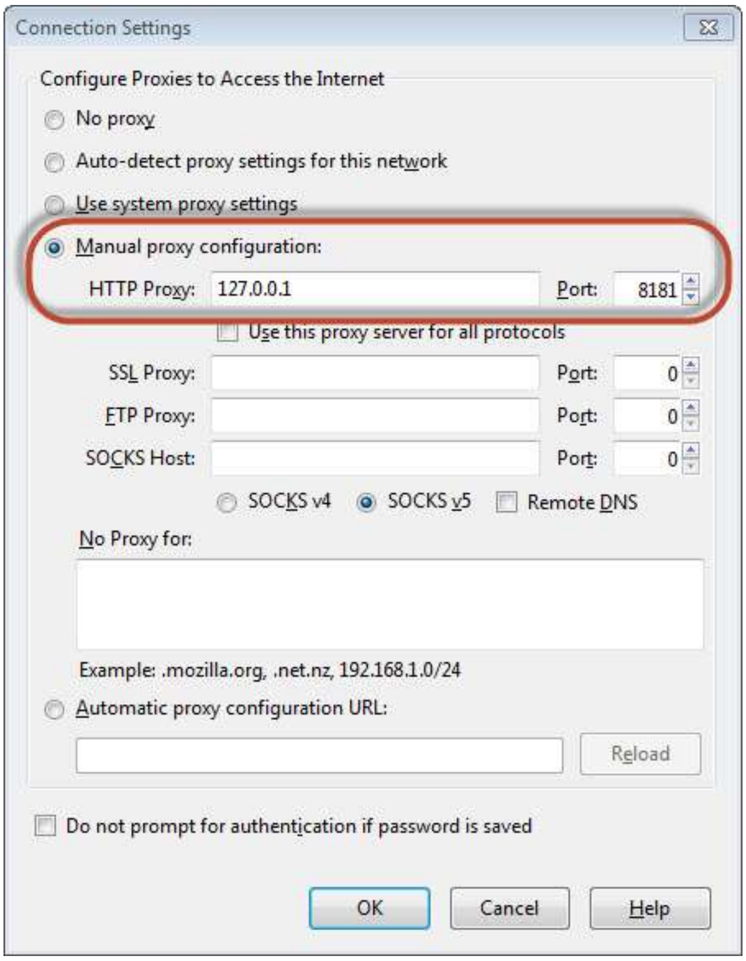
Target Scope

Define the in-scope targets for your current work. This configuration affects the behavior of tools throughout the suite. All fields take regex strings.

Include in scope

Enabled	Protocol	Host / IP range	Port	File
☐	Any			
☐	~~Any~~	~~itelearn~~		
☑	Any	localhost	8080	

3. Then make your browser proxy settings to listen to the port 8181 (Burp Suite port). Thus we have configured the Web proxy to intercept the traffic between the client (browser) and the server (Webserver) as shown below:



Configure Proxies to Access the Internet

- ○ No proxy
- ○ Auto-detect proxy settings for this network
- ○ Use system proxy settings
- ● Manual proxy configuration:

HTTP Proxy: 127.0.0.1 Port: 8181

☐ Use this proxy server for all protocols

SSL Proxy: Port: 0

FTP Proxy: Port: 0

SOCKS Host: Port: 0

○ SOCKS v4 ● SOCKS v5 ☐ Remote DNS

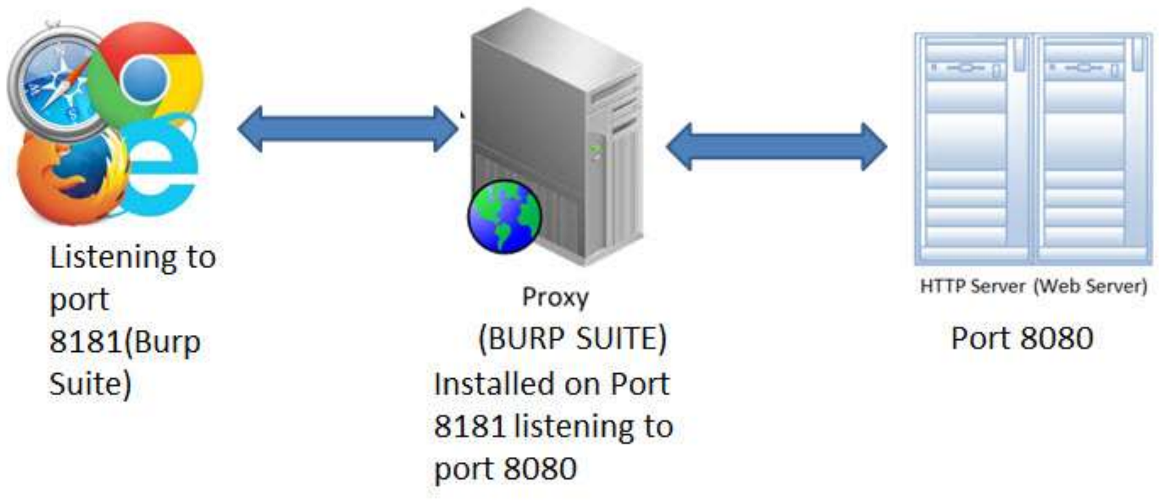
No Proxy for:

Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

☐ Do not prompt for authentication if password is saved

4. The snapshot of the configuration is shown below with a help of a simple workflow diagram as shown below


Listening to port 8181(Burp Suite)
Proxy
(BURP SUITE)
Installed on Port 8181 listening to port 8080
HTTP Server (Web Server)
Port 8080

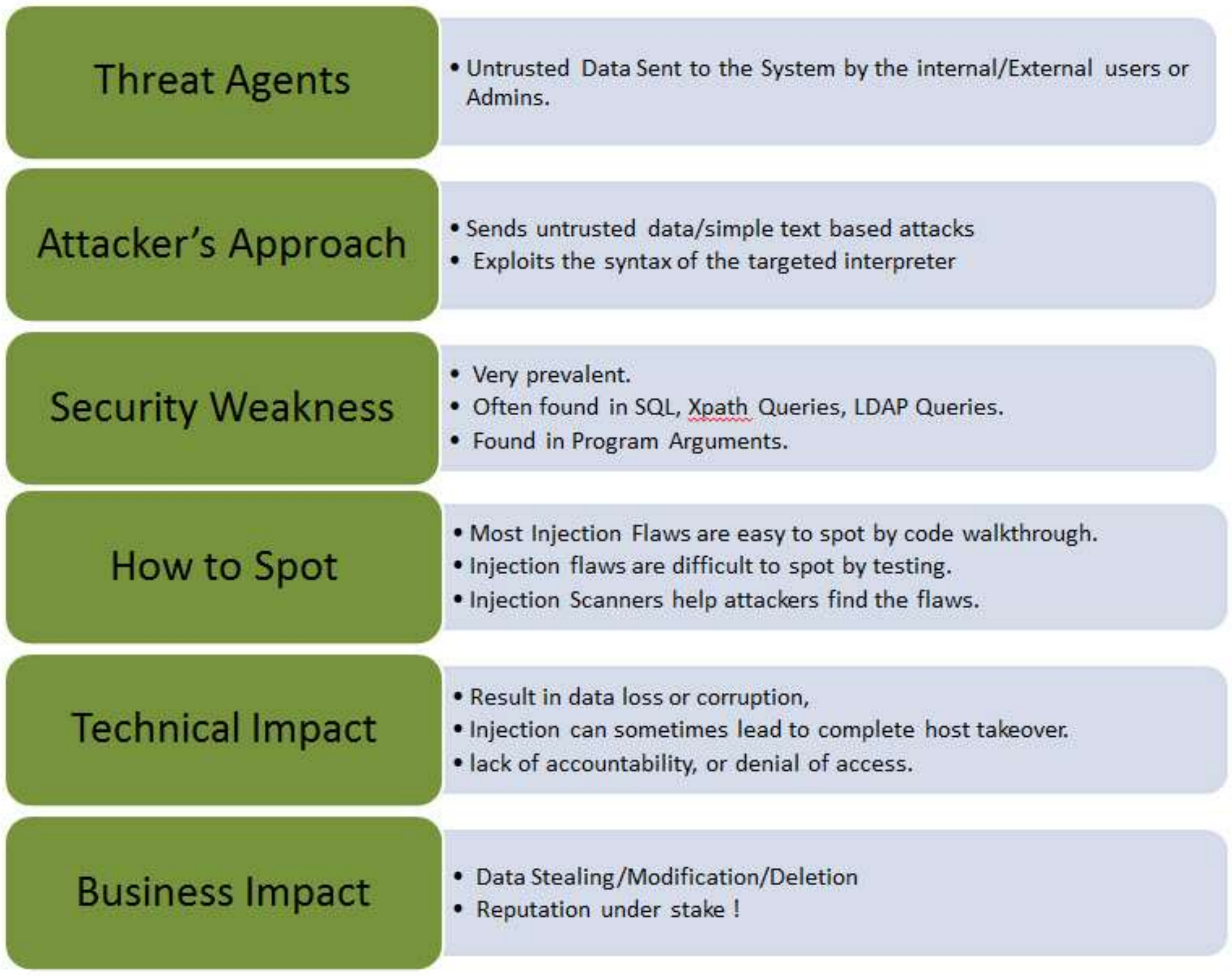
11. Testing Injection

Injection technique consists of injecting a SQL query or a command using the input fields of the application.

Web Application - Injection

A successful SQL injection can read, modify sensitive data from the database, and can also delete data from a database. It also enables the hacker to perform administrative operations on the database such as shutdown the DBMS/dropping databases.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Untrusted Data Sent to the System by the internal/External users or Admins.
Attacker's Approach	• Sends untrusted data/simple text based attacks • Exploits the syntax of the targeted interpreter
Security Weakness	• Very prevalent. • Often found in SQL, Xpath Queries, LDAP Queries. • Found in Program Arguments.
How to Spot	• Most Injection Flaws are easy to spot by code walkthrough. • Injection flaws are difficult to spot by testing. • Injection Scanners help attackers find the flaws.
Technical Impact	• Result in data loss or corruption, • Injection can sometimes lead to complete host takeover. • lack of accountability, or denial of access.
Business Impact	• Data Stealing/Modification/Deletion • Reputation under stake !

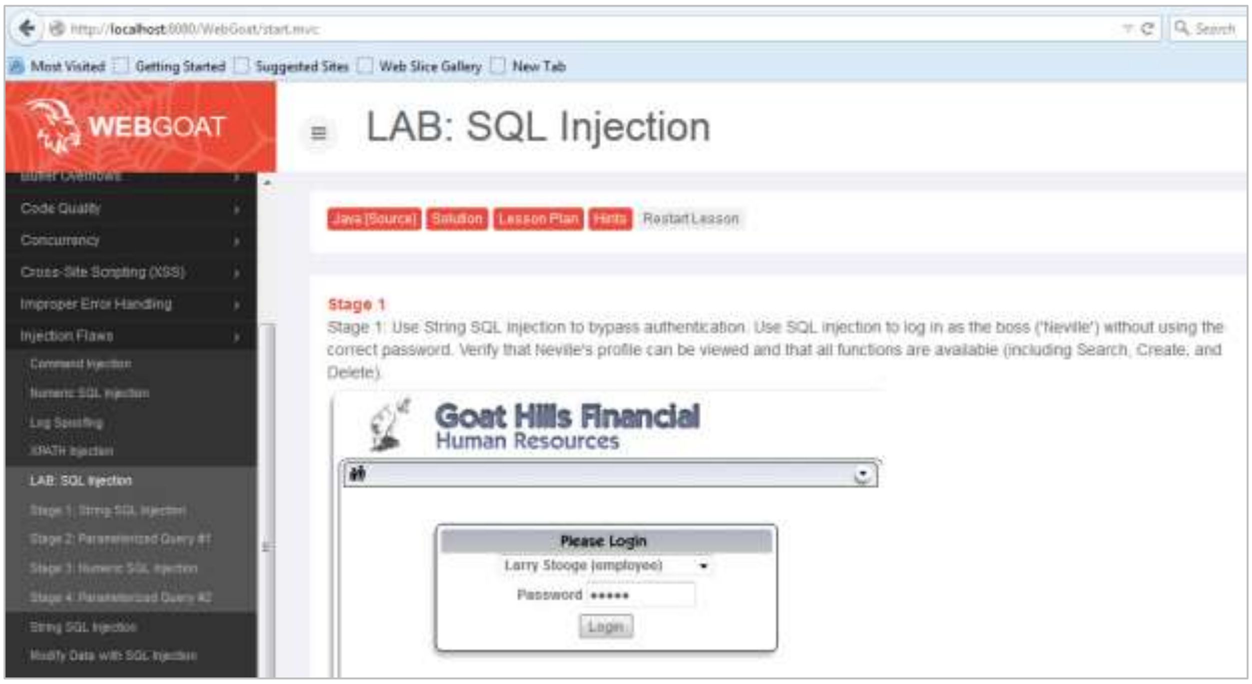
Examples

The application uses untrusted data in the construction of the following vulnerable SQL call:

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

Hands On

1. Navigate to the SQL Injection area of the application as shown below.



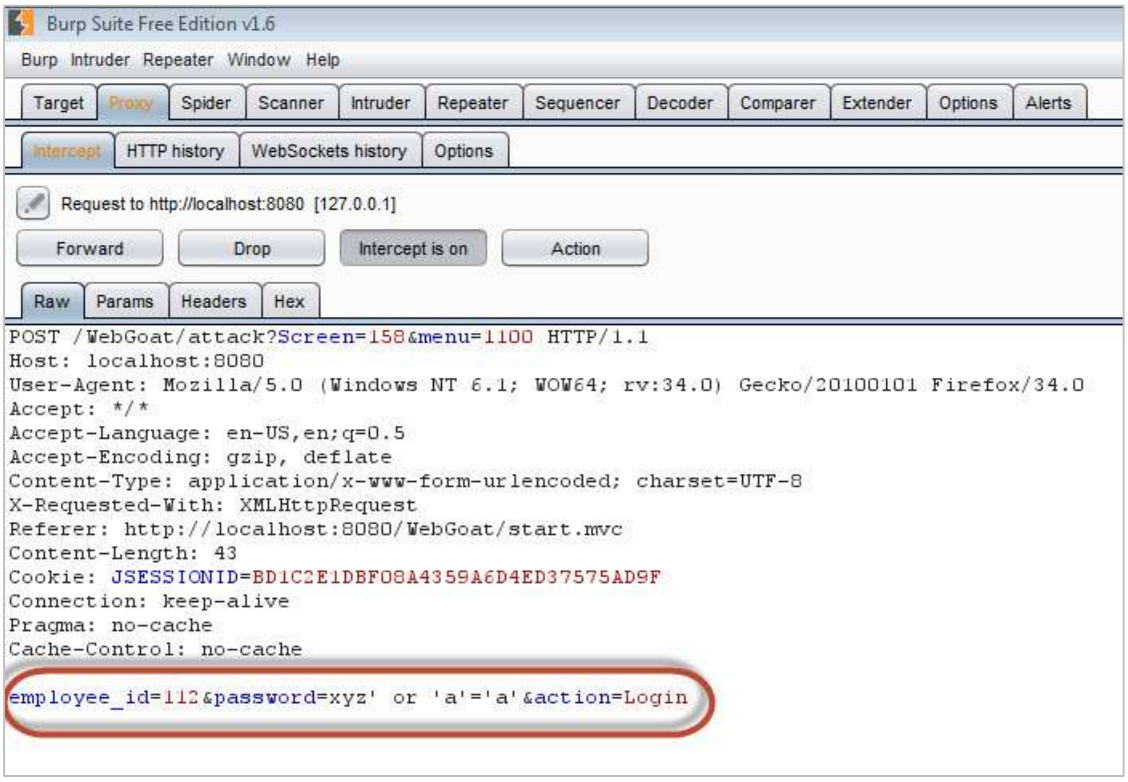
LAB: SQL Injection

Stage 1

Stage 1: Use String SQL injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete).

2. As given in the exercise, we use String SQL Injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete).

3. We will Inject a SQL such that we are able to bypass the password by sending the parameter as 'a'='a' or 1=1

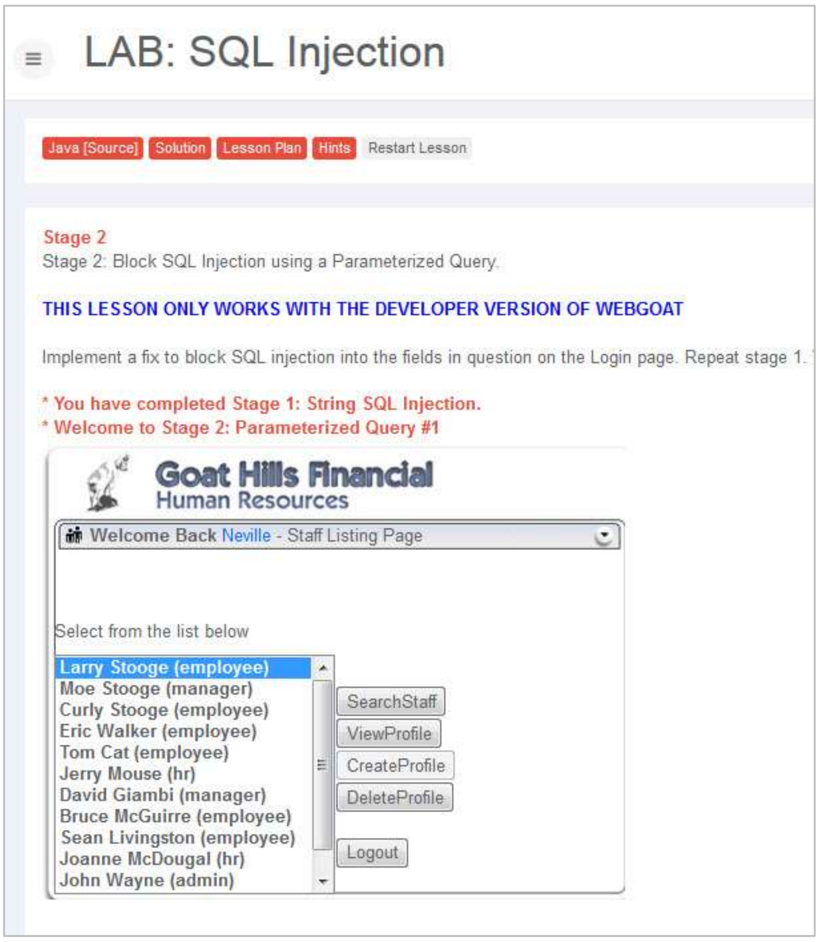

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Params Headers Hex

Request to http://localhost:8080 [127.0.0.1]

```
POST /WebGoat/attack?Screen=158&menu=1100 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Length: 43
Cookie: JSESSIONID=BD1C2E1DBF08A4359A6D4ED37575AD9F
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

employee_id=112&password=xyz' or 'a'='a'&action=Login
```

4. Post Exploitation, we are able to login as Neville who is the Admin as shown below.



LAB: SQL Injection

Stage 2
Stage 2: Block SQL Injection using a Parameterized Query.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block SQL injection into the fields in question on the Login page. Repeat stage 1.

* You have completed Stage 1: String SQL Injection.
* Welcome to Stage 2: Parameterized Query #1

Welcome Back Neville - Staff Listing Page

Select from the list below

Larry Stooge (employee)
Moe Stooge (manager)
Curly Stooge (employee)
Eric Walker (employee)
Tom Cat (employee)
Jerry Mouse (hr)
David Giambi (manager)
Bruce McGuirre (employee)
Sean Livingston (employee)
Joanne McDougal (hr)
John Wayne (admin)

Preventing SQL Injection

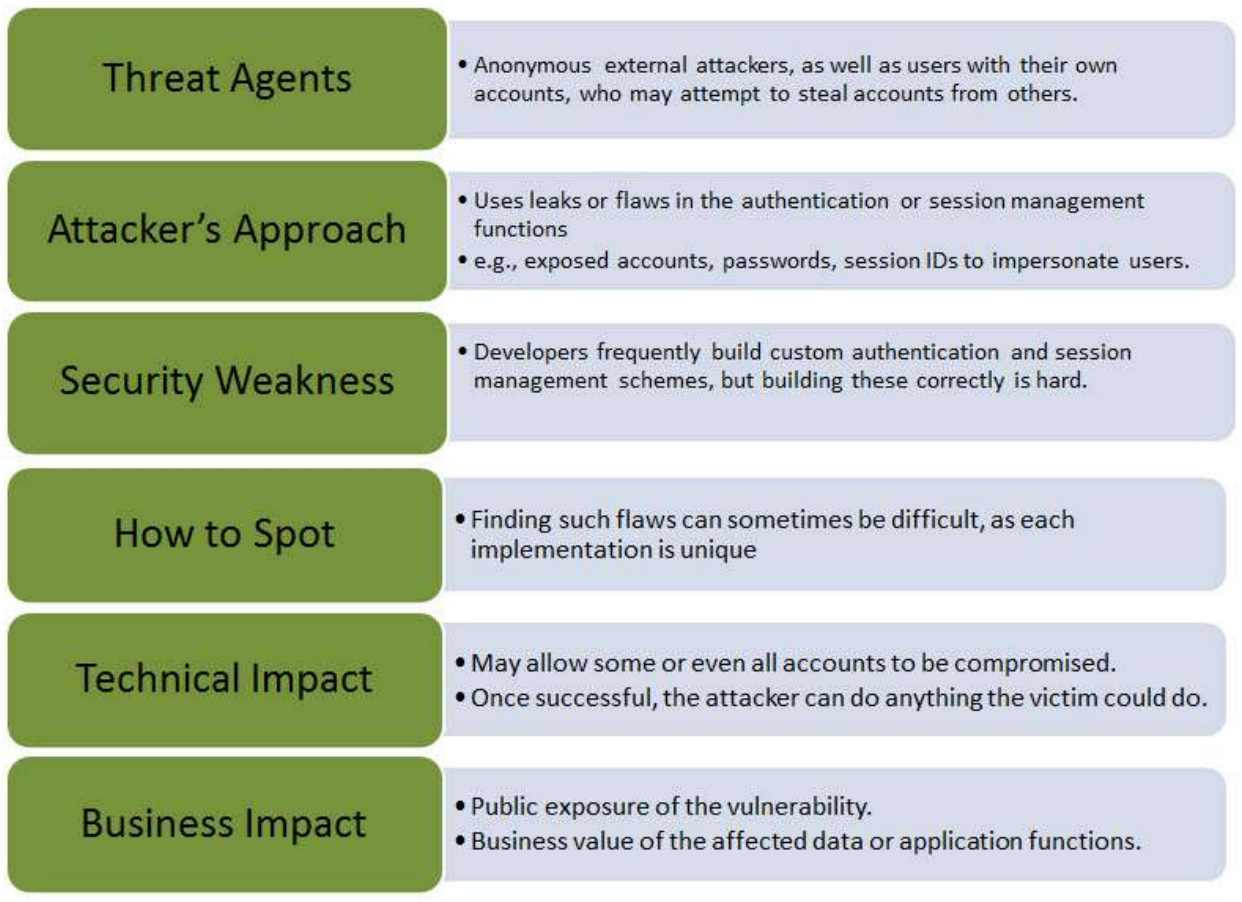
There are plenty of ways to prevent SQL injection. When developers write the code, they should ensure that they handle special characters accordingly. There are cheat sheets/prevention techniques available from OWASP which is definitely a guide for developers.

- Using Parameterized Queries
- Escaping all User Supplied Input
- Enable Least Privilege for the database for the end users

12. Testing Broken Authentication

When authentication functions related to the application are not implemented correctly, it allows hackers to compromise passwords or session ID's or to exploit other implementation flaws using other users credentials.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others.
Attacker's Approach	• Uses leaks or flaws in the authentication or session management functions • e.g., exposed accounts, passwords, session IDs to impersonate users.
Security Weakness	• Developers frequently build custom authentication and session management schemes, but building these correctly is hard.
How to Spot	• Finding such flaws can sometimes be difficult, as each implementation is unique
Technical Impact	• May allow some or even all accounts to be compromised. • Once successful, the attacker can do anything the victim could do.
Business Impact	• Public exposure of the vulnerability. • Business value of the affected data or application functions.

Example

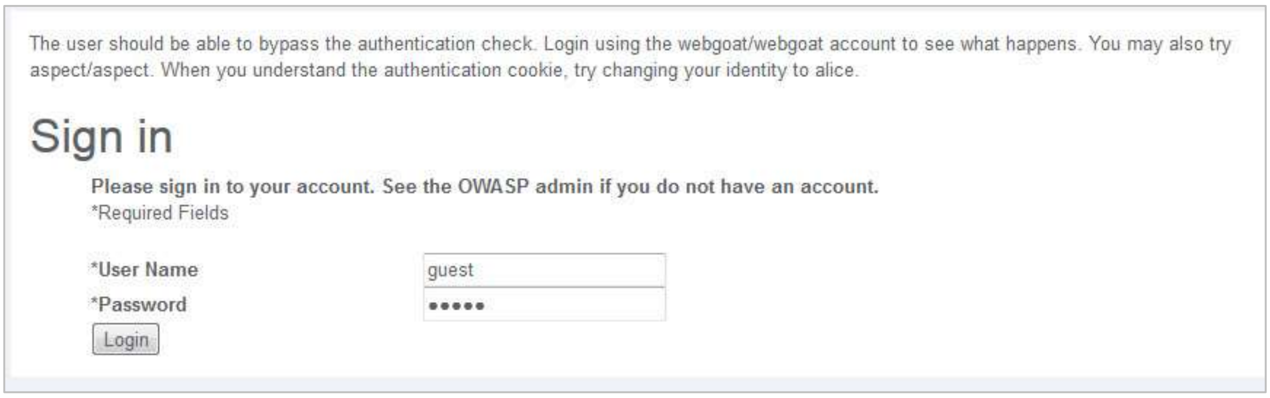
```
An e-commerce application supports URL rewriting, putting session IDs in the URL:


http://example.com/sale/saleitems/jsessionid=2P0OC2JSNDLPSKHCJUN2JV/?item=laptop
```

An authenticated user of the site forwards the URL to their friends to know about the discounted sales. He e-mails the above link without knowing that the user is also giving away the session IDs. When his friends use the link, they use his session and credit card.

Hands ON

1. Login to Webgoat and navigate to 'Session Management Flaws' Section. Let us bypass the authetication by spoofing the cookie. Below is the snapshot of the scenario.



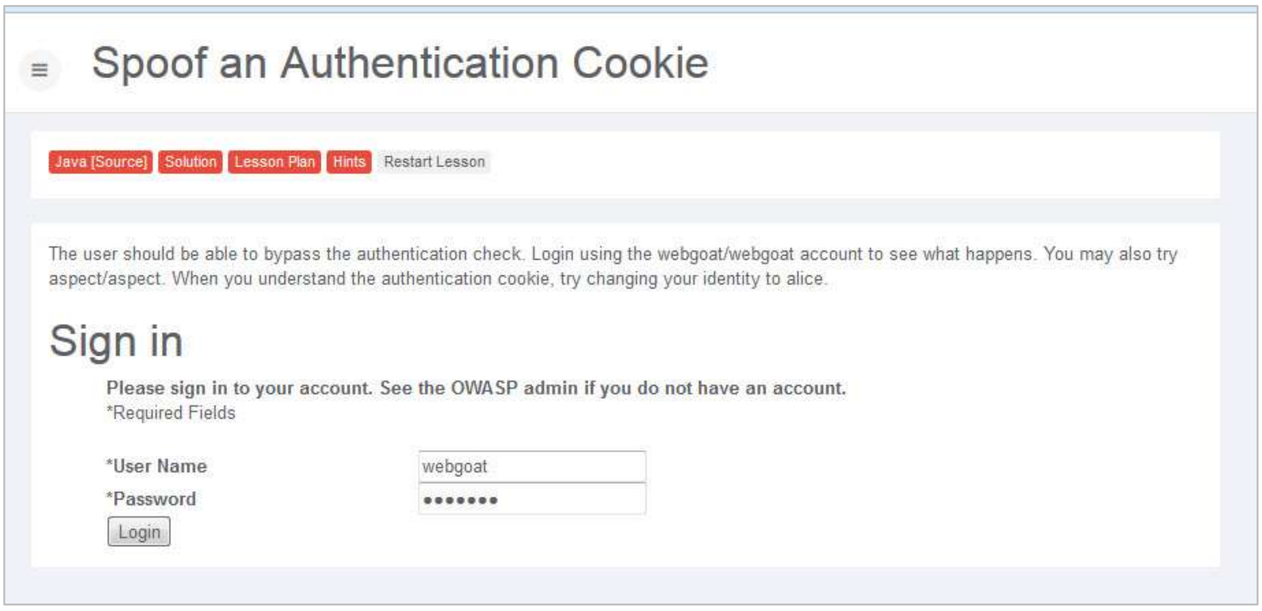
The user should be able to bypass the authentication check. Login using the webgoat/webgoat account to see what happens. You may also try aspect/aspect. When you understand the authentication cookie, try changing your identity to alice.

Sign in

Please sign in to your account. See the OWASP admin if you do not have an account.
*Required Fields

*User Name: guest
*Password: •••••

2. When we login using the credentials webgoat/webgoat, we find from Burp Suite that the JSESSION ID is C8F3177CCAFF380441ABF71090748F2E while the AuthCookie=65432ubphcfx upon successful authentication.



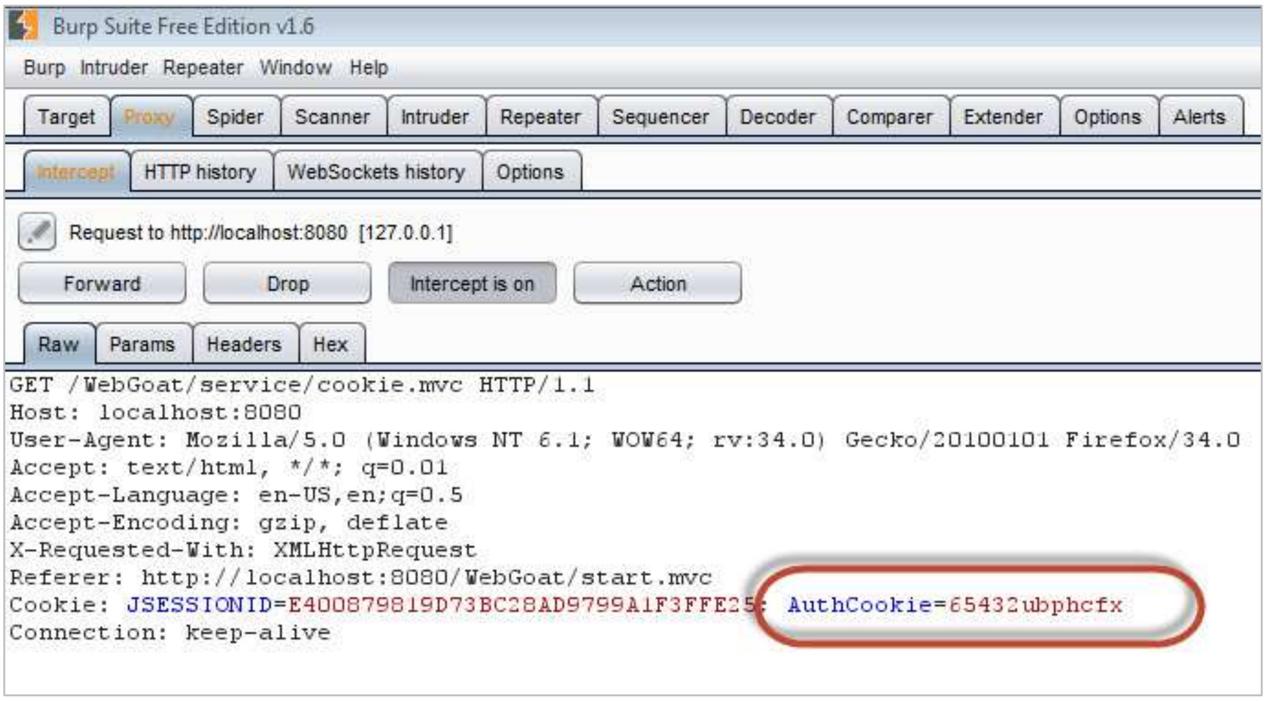
Spoof an Authentication Cookie

The user should be able to bypass the authentication check. Login using the webgoat/webgoat account to see what happens. You may also try aspect/aspect. When you understand the authentication cookie, try changing your identity to alice.

Sign in

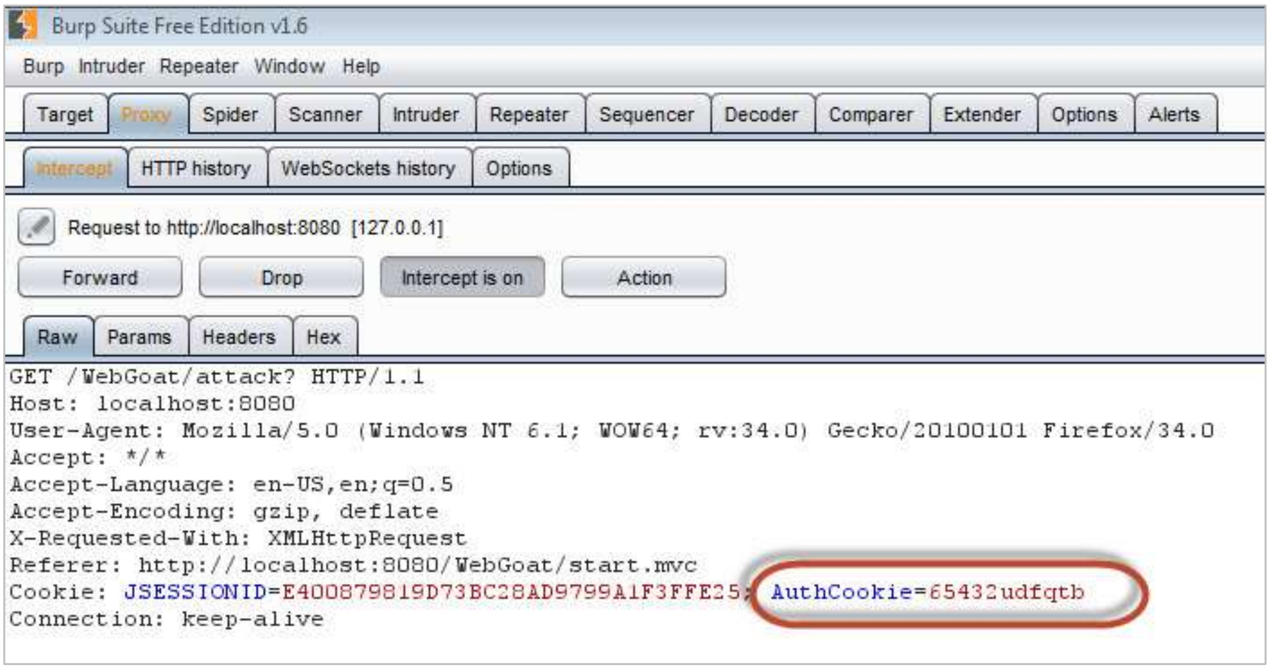
Please sign in to your account. See the OWASP admin if you do not have an account.
*Required Fields

*User Name: webgoat
*Password: •••••••


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
GET /WebGoat/service/cookie.mvc HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: text/html, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=E400879819D73BC28AD9799A1F3FFE25; AuthCookie=65432ubphcfx
Connection: keep-alive
```

3. When we login using the credentials aspect/aspect, we find from Burp Suite that the JSESSION ID is C8F3177CCAFF380441ABF71090748F2E while the AuthCookie=65432udfqtb upon successful authentication.


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex

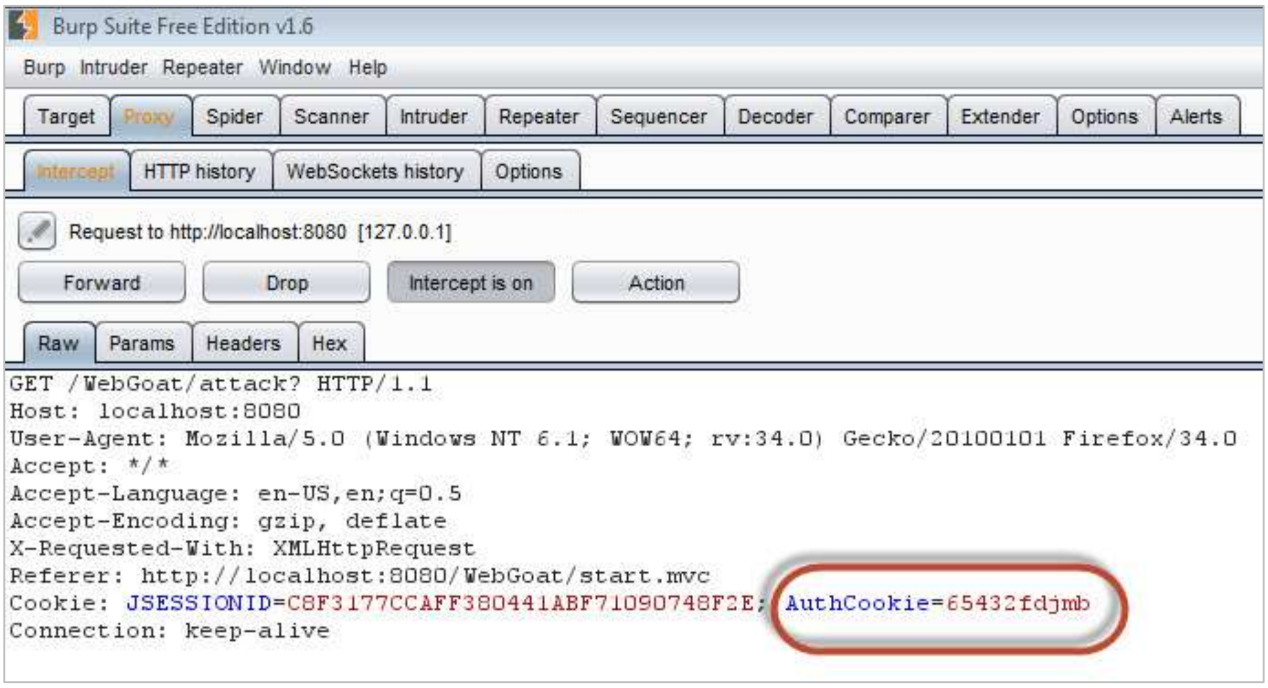
```
GET /WebGoat/attack? HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=E400879819D73BC28AD9799A1F3FFE25; AuthCookie=65432udfqtb
Connection: keep-alive
```

4. Now we need to analyze the AuthCookie Patterns. The first half '65432' is common for both authentications. Hence we are now interested in analyzing the last part of the authcookie values such as - ubphcfx for webgoat user and udfqtb for aspect user respectively.

5. If we take a deep look at the AuthCookie values, the last part is having the same length as that of user name. Hence it is evident that the username is used with some encryption method. Upon trial and errors/brute force mechanisms, we find that after reversing the

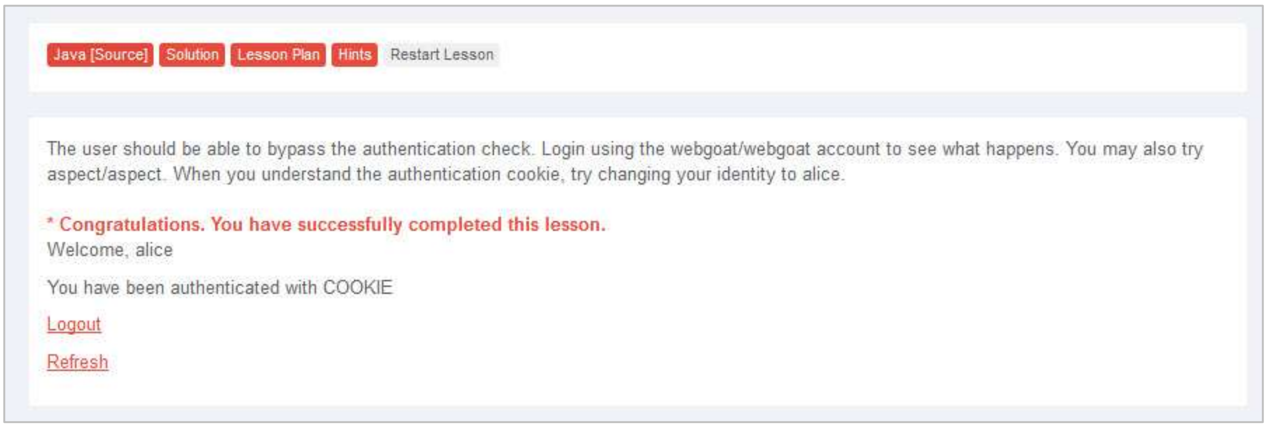
user name, webgoat; we end up with taogbew and then the before alphabet character is what being used as AuthCookie. i.e ubphcfx.

6. If we pass this cookie value and let us see what happens. Upon authenticating as user webgoat, change the AuthCookie value to mock the user Alice by finding the AuthCookie for the same by performing step#4 and step#5.


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Params Headers Hex

Request to http://localhost:8080 [127.0.0.1]

```
GET /WebGoat/attack? HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=C8F3177CCAFF380441ABF71090748F2E; AuthCookie=65432fdjmb
Connection: keep-alive
```



The user should be able to bypass the authentication check. Login using the webgoat/webgoat account to see what happens. You may also try aspect/aspect. When you understand the authentication cookie, try changing your identity to alice.

* Congratulations. You have successfully completed this lesson.
Welcome, alice

You have been authenticated with COOKIE

Logout

Refresh

Preventing Mechanisms

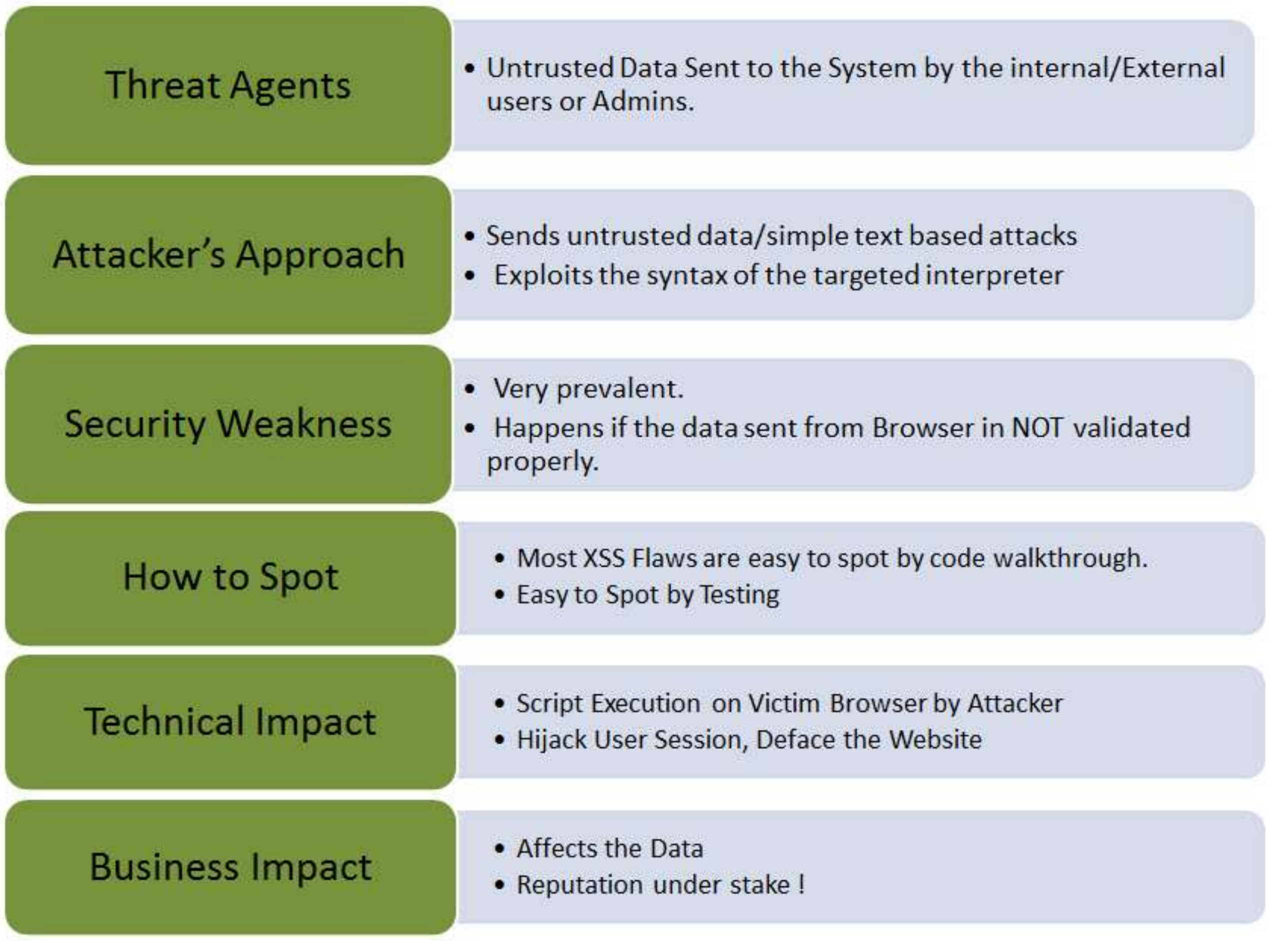
- Develop a strong authentication and session management controls such that it meets all the authentication and session management requirements defined in OWASP's Application Security Verification Standard.

- Developers should ensure that they avoid XSS flaws that can be used to steal session IDs.

13. Tesing Cross-site Scripting

Cross-site Scripting (XSS) happens whenever an application takes untrusted data and sends it to the client (browser) without validation. This allows attackers to execute malicious scripts in the victim's browser which can result in user sessions hijack, defacing web sites or redirect the user to malicious sites.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Untrusted Data Sent to the System by the internal/External users or Admins.
Attacker's Approach	• Sends untrusted data/simple text based attacks • Exploits the syntax of the targeted interpreter
Security Weakness	• Very prevalent. • Happens if the data sent from Browser in NOT validated properly.
How to Spot	• Most XSS Flaws are easy to spot by code walkthrough. • Easy to Spot by Testing
Technical Impact	• Script Execution on Victim Browser by Attacker • Hijack User Session, Deface the Website
Business Impact	• Affects the Data • Reputation under stake !

Types of XSS

- **Stored XSS -** Stored XSS also known as persistent XSS occurs when user input is stored on the target server such as database/message forum/comment field etc. Then the victim is able to retrieve the stored data from the web application.

- **Reflected XSS -** Reflected XSS also known as non-persistent XSS occurs when user input is immediately returned by a web application in an error message/search result or the input provided by the user as part of the request and without permanently storing the user provided data.

- **DOM Based XSS -** DOM Based XSS is a form of XSS when the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser.

Example

The application uses untrusted data in the construction without validation. The special characters ought to be escaped.

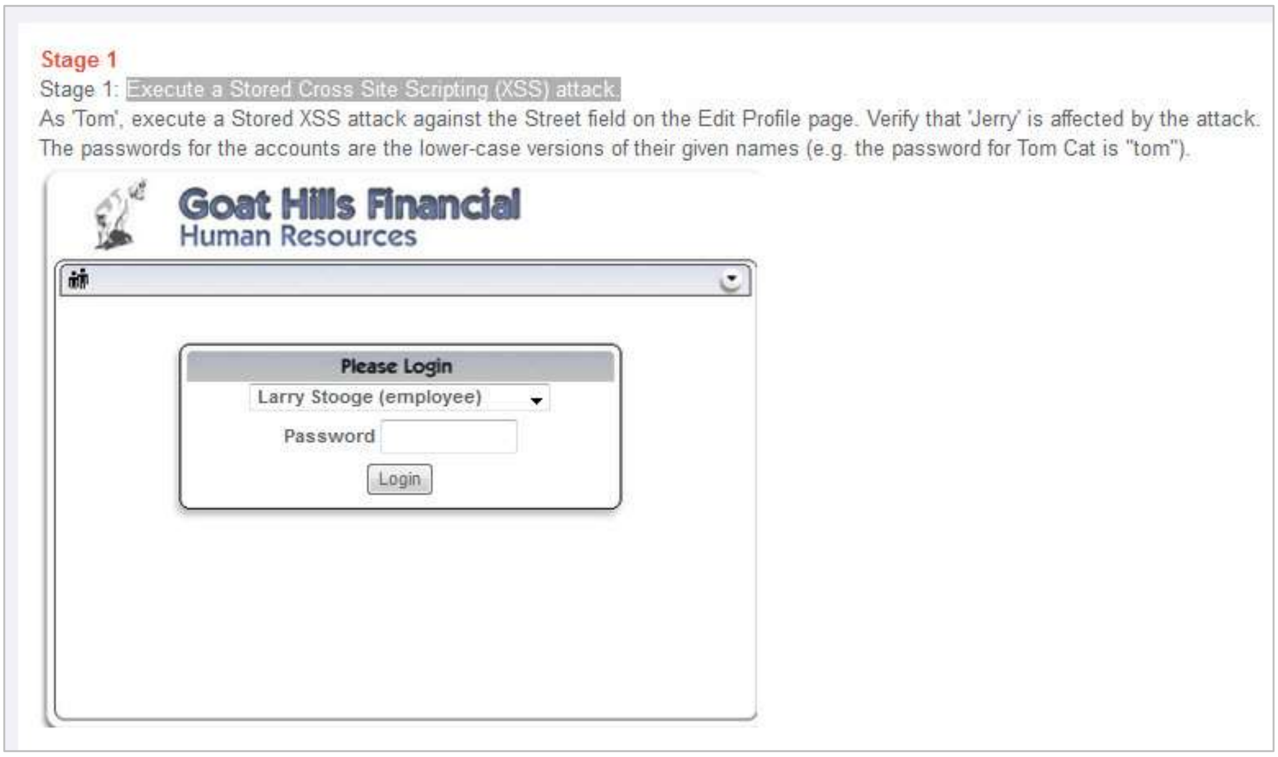
```
http://www.webpage.org/task/Rule1?query=try
```

The attacker modifies the query parameter in their browser to:

```
http://www.webpage.org/task/Rule1?query=<h3>Hello from XSS"</h3>
```

Hands ON

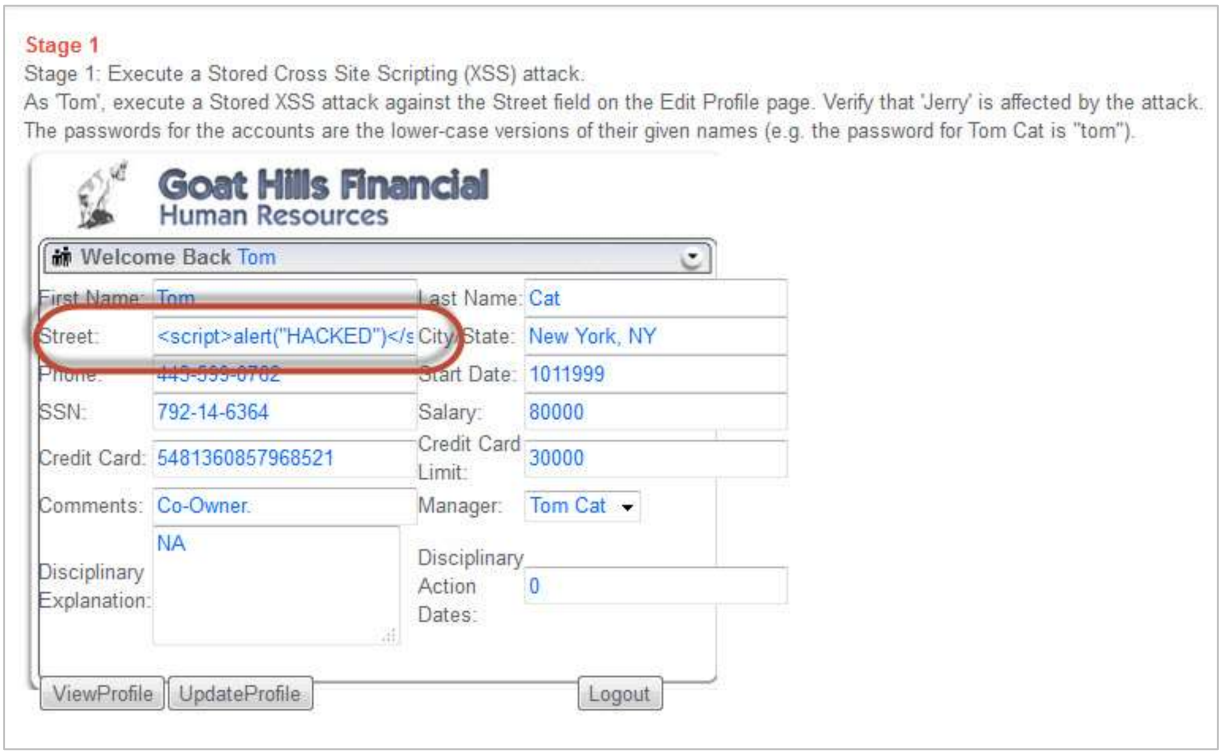
1. Login to Webgoat and navigate to cross-site scripting (XSS) Section. Let us execute a Stored Cross-site Scripting (XSS) attack. Below is the snapshot of the scenario.



Stage 1
Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.
As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.
The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").

2. As per the scenario, let us login as Tom with password 'tom' as mentioned in the scenario itself. Click 'view profile' and get into edit mode. Since tom is the attacker, let us inject Java script into those edit boxes.

```
<script> alert("HACKED")</script>
```



Stage 1

Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.

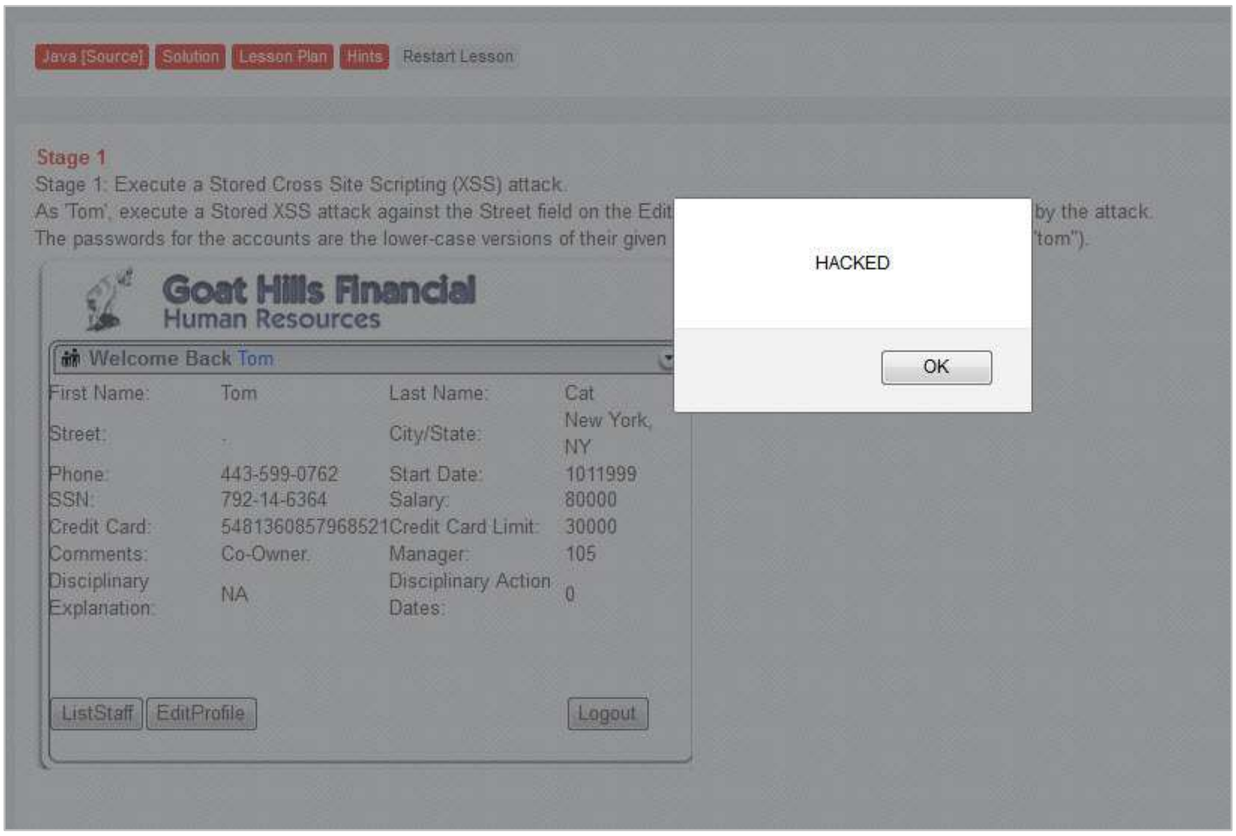
The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").

Goat Hills Financial
Human Resources

Welcome Back Tom

First Name:	Tom	Last Name:	Cat
Street:	<script>alert("HACKED")</s	City/State:	New York, NY
Phone:	443-599-0762	Start Date:	1011999
SSN:	792-14-6364	Salary:	80000
Credit Card:	5481360857968521	Credit Card Limit:	30000
Comments:	Co-Owner.	Manager:	Tom Cat
Disciplinary Explanation:	NA	Disciplinary Action Dates:	0

3. As soon as the update is over, tom receives an alert box with the message "hacked" which means that the app is vulnerable.



Stage 1

Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit by the attack.

The passwords for the accounts are the lower-case versions of their given 'tom").

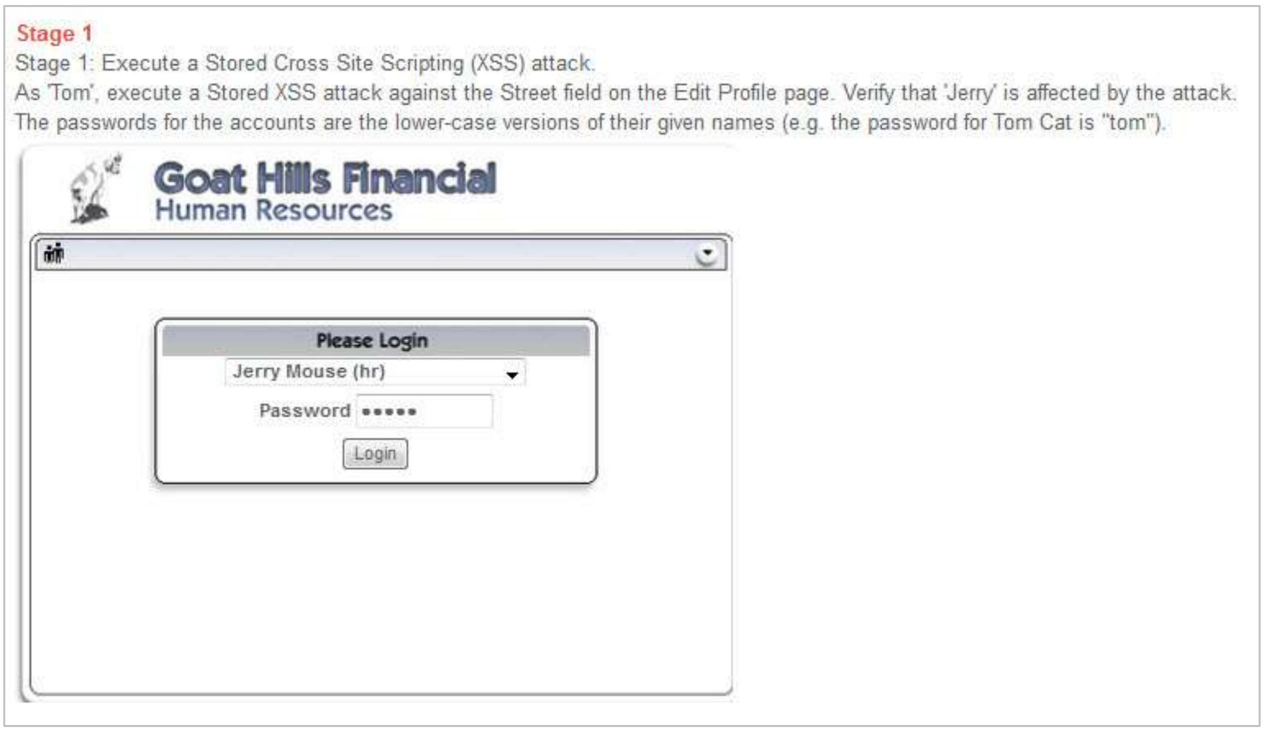
HACKED

Goat Hills Financial
Human Resources

Welcome Back Tom

First Name:	Tom	Last Name:	Cat
Street:	.	City/State:	New York, NY
Phone:	443-599-0762	Start Date:	1011999
SSN:	792-14-6364	Salary:	80000
Credit Card:	5481360857968521	Credit Card Limit:	30000
Comments:	Co-Owner.	Manager:	105
Disciplinary Explanation:	NA	Disciplinary Action Dates:	0

4. Now as per the scenario, we need to login as jerry (HR) and check if jerry is affected by the injected script.



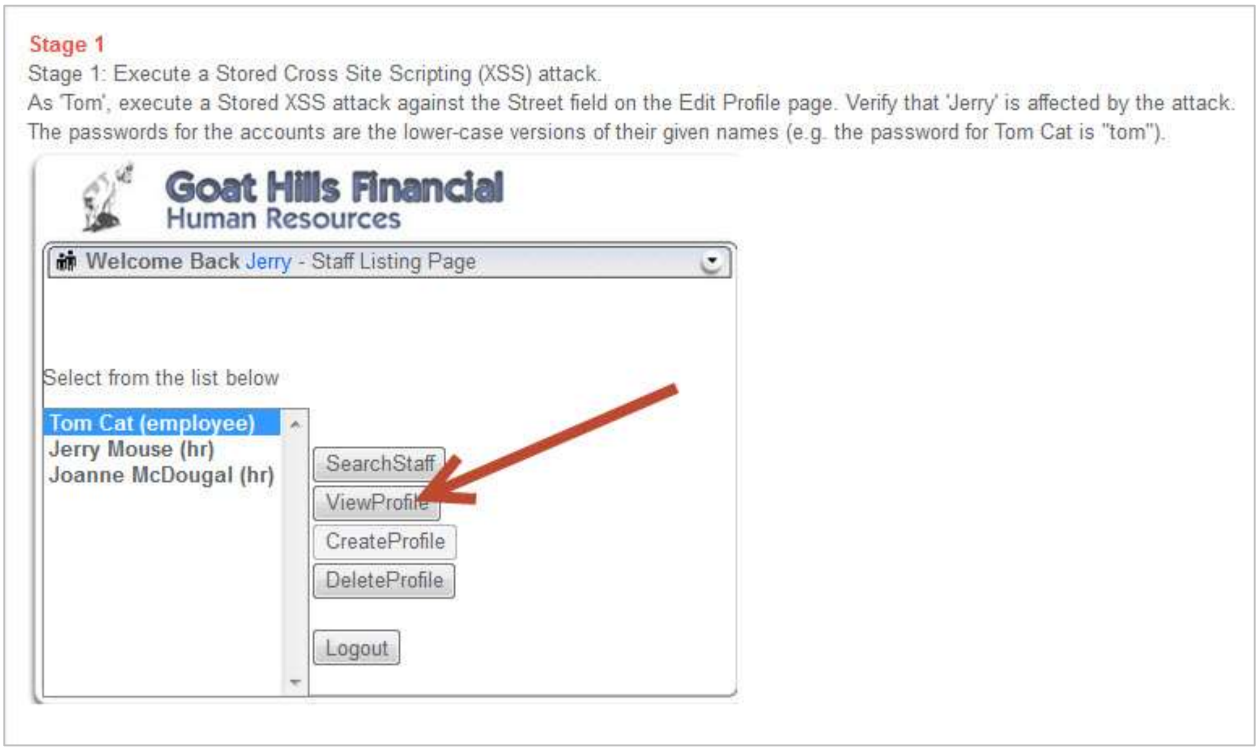
Stage 1

Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.

The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").

5. After logging in as Jerry, select 'Tom' and click 'view profile' as shown below.



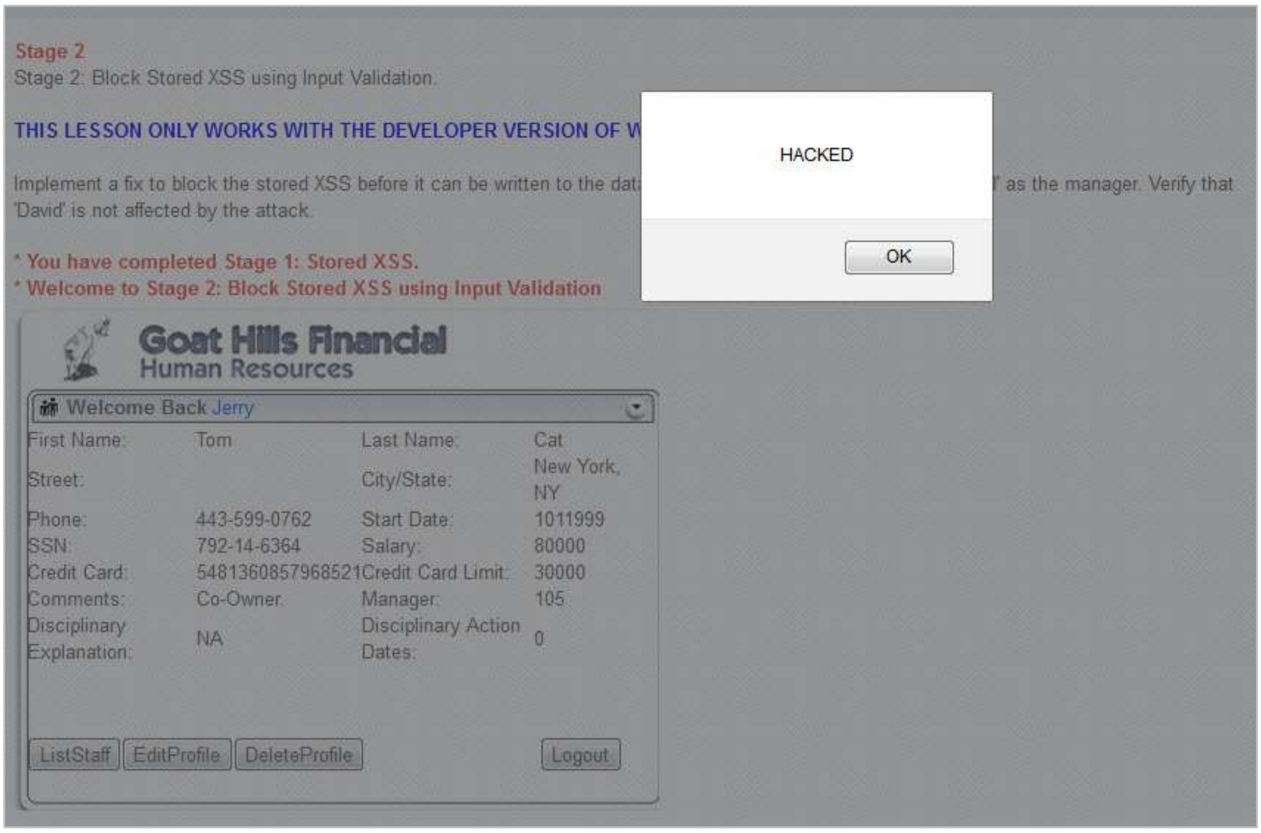
Stage 1

Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.

The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").

While viewing tom's profile from Jerry's account, he is able to get the same message box.


Stage 2
Stage 2: Block Stored XSS using Input Validation.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF W

Implement a fix to block the stored XSS before it can be written to the dat 'T' as the manager. Verify that 'David' is not affected by the attack.

* You have completed Stage 1: Stored XSS.
* Welcome to Stage 2: Block Stored XSS using Input Validation

HACKED

Goat Hills Financial
Human Resources

Welcome Back Jerry

First Name:	Tom	Last Name:	Cat
Street:		City/State:	New York, NY
Phone:	443-599-0762	Start Date:	1011999
SSN:	792-14-6364	Salary:	80000
Credit Card:	5481360857968521	Credit Card Limit:	30000
Comments:	Co-Owner.	Manager:	105
Disciplinary Explanation:	NA	Disciplinary Action Dates:	0

6. This message box is just an example, but the actual attacker can perform much more than just displaying a message box.

Preventive Mechanisms

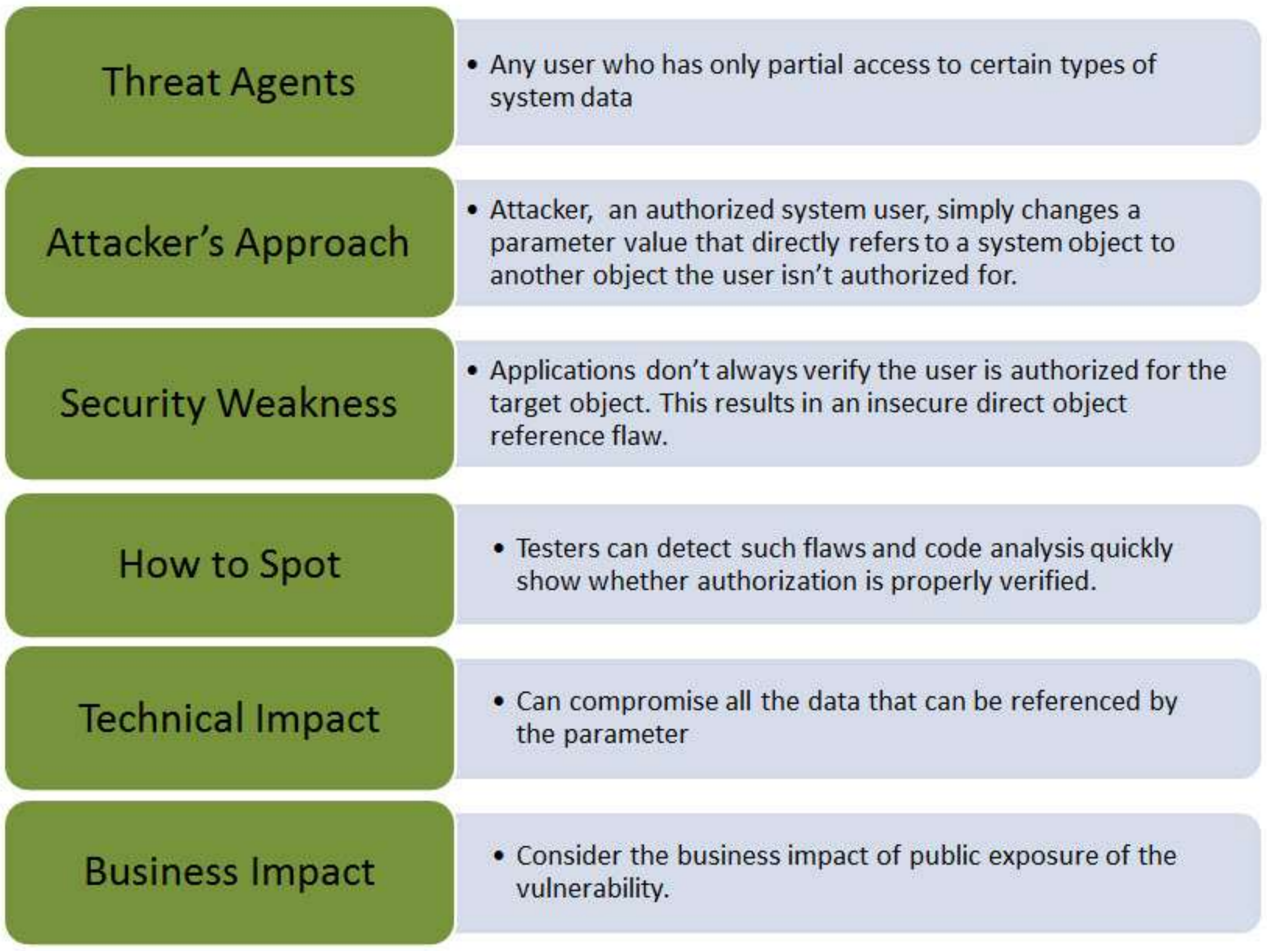
- Developers have to ensure that they escape all untrusted data based on the HTML context such as body, attribute, JavaScript, CSS, or URL that the data is placed into.

- For those applications that need special characters as input, there should be robust validation mechanisms in place before accepting them as valid inputs.

14. Insecure Direct Object References

A direct object reference is likely to occur when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key without any validation mechanism which allows attackers to manipulate these references to access unauthorized data.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Any user who has only partial access to certain types of system data
Attacker's Approach	• Attacker, an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for.
Security Weakness	• Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw.
How to Spot	• Testers can detect such flaws and code analysis quickly show whether authorization is properly verified.
Technical Impact	• Can compromise all the data that can be referenced by the parameter
Business Impact	• Consider the business impact of public exposure of the vulnerability.

Example

The App uses unverified data in a SQL call that is accessing account information.

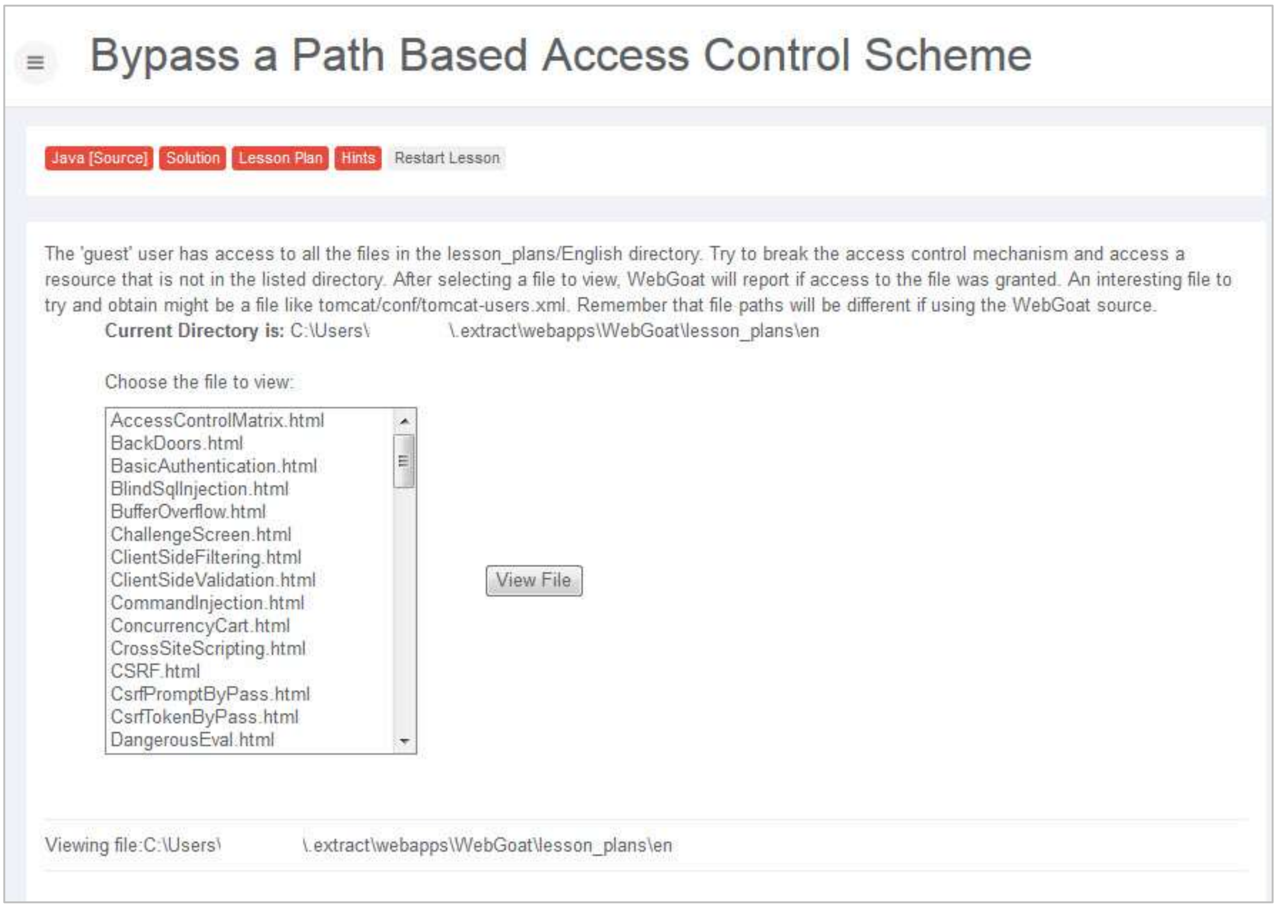
```
String sqlquery = "SELECT * FROM useraccounts WHERE account = ?";
PreparedStatement st = connection.prepareStatement(sqlquery, �);
st.setString( 1, request.getParameter("acct"));
ResultSet results = st.executeQuery( );
```

The attacker modifies the query parameter in their browser to point to Admin.

```
http://webapp.com/app/accountInfo?acct=admin
```

Hands ON

1. Login to Webgoat and navigate to access control flaws Section. The goal is to retrieve the tomcat-users.xml by navigating to the path where it is located. Below is the snapshot of the scenario.


Bypass a Path Based Access Control Scheme

The 'guest' user has access to all the files in the lesson_plans/English directory. Try to break the access control mechanism and access a resource that is not in the listed directory. After selecting a file to view, WebGoat will report if access to the file was granted. An interesting file to try and obtain might be a file like tomcat/conf/tomcat-users.xml. Remember that file paths will be different if using the WebGoat source.

Current Directory is: C:\Users\ \.extract\webapps\WebGoat\lesson_plans\en

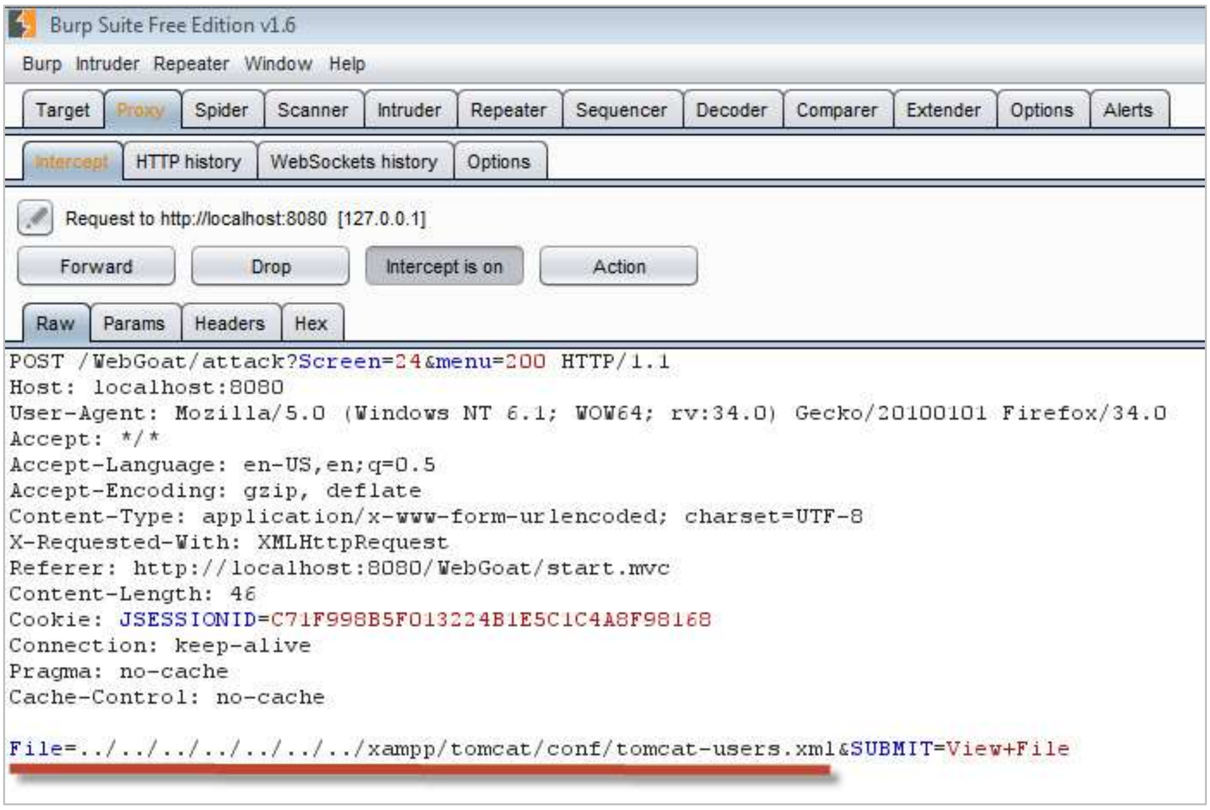
Choose the file to view:

AccessControlMatrix.html
BackDoors.html
BasicAuthentication.html
BlindSqlInjection.html
BufferOverflow.html
ChallengeScreen.html
ClientSideFiltering.html
ClientSideValidation.html
CommandInjection.html
ConcurrencyCart.html
CrossSiteScripting.html
CSRF.html
CsrfPromptByPass.html
CsrfTokenByPass.html
DangerousEval.html

Viewing file:C:\Users\ \.extract\webapps\WebGoat\lesson_plans\en

2. The path of the file is displayed in 'the current directory is' field - C:\Users\userName$\.extract\webapps\WebGoat\lesson_plans\en and we also know that the tomcat-users.xml file is kept under C:\xampp\tomcat\conf

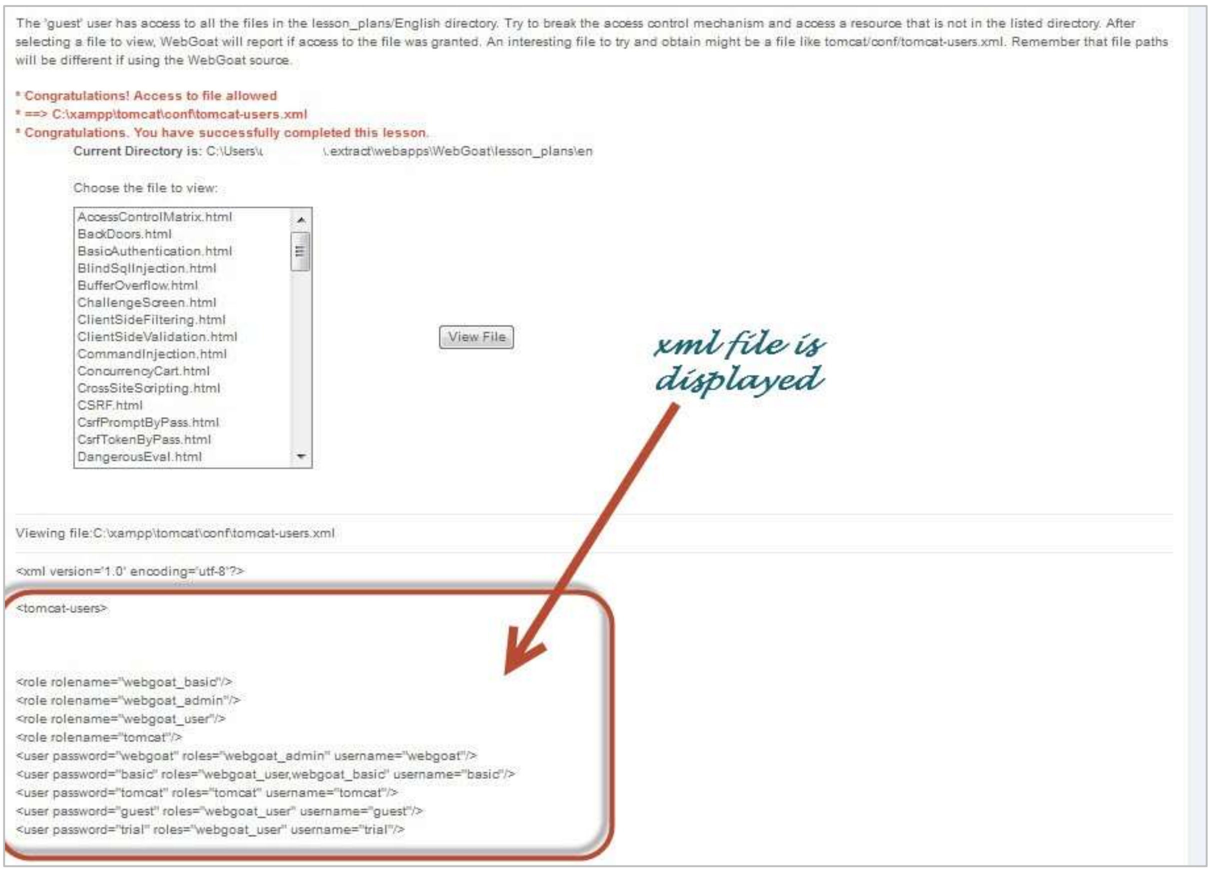
3. We need to traverse all the way out of the current directory and navigate from C:\ Drive. We can perform the same by intercepting the traffic using Burp Suite.


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=24&menu=200 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Length: 46
Cookie: JSESSIONID=C71F998B5F013224B1E5C1C4A8F98168
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

File=../../../../../../../xampp/tomcat/conf/tomcat-users.xml&SUBMIT=View+File
```

4. If the attempt is successful, it displays the tomcat-users.xml with the message "Congratulations. You have successfully completed this lesson."



The 'guest' user has access to all the files in the lesson_plans/English directory. Try to break the access control mechanism and access a resource that is not in the listed directory. After selecting a file to view, WebGoat will report if access to the file was granted. An interesting file to try and obtain might be a file like tomcat/conf/tomcat-users.xml. Remember that file paths will be different if using the WebGoat source.

* Congratulations! Access to file allowed
* ==> C:\xampp\tomcat\conf\tomcat-users.xml
* Congratulations. You have successfully completed this lesson.

Current Directory is: C:\Users\ \.extract\webapps\WebGoat\lesson_plans\en

Choose the file to view:

Viewing file:C:\xampp\tomcat\conf\tomcat-users.xml

```
<xml version='1.0' encoding='utf-8'?>
<tomcat-users>


<role rolename="webgoat_basic"/>
<role rolename="webgoat_admin"/>
<role rolename="webgoat_user"/>
<role rolename="tomcat"/>
<user password="webgoat" roles="webgoat_admin" username="webgoat"/>
<user password="basic" roles="webgoat_user,webgoat_basic" username="basic"/>
<user password="tomcat" roles="tomcat" username="tomcat"/>
<user password="guest" roles="webgoat_user" username="guest"/>
<user password="trial" roles="webgoat_user" username="trial"/>
```

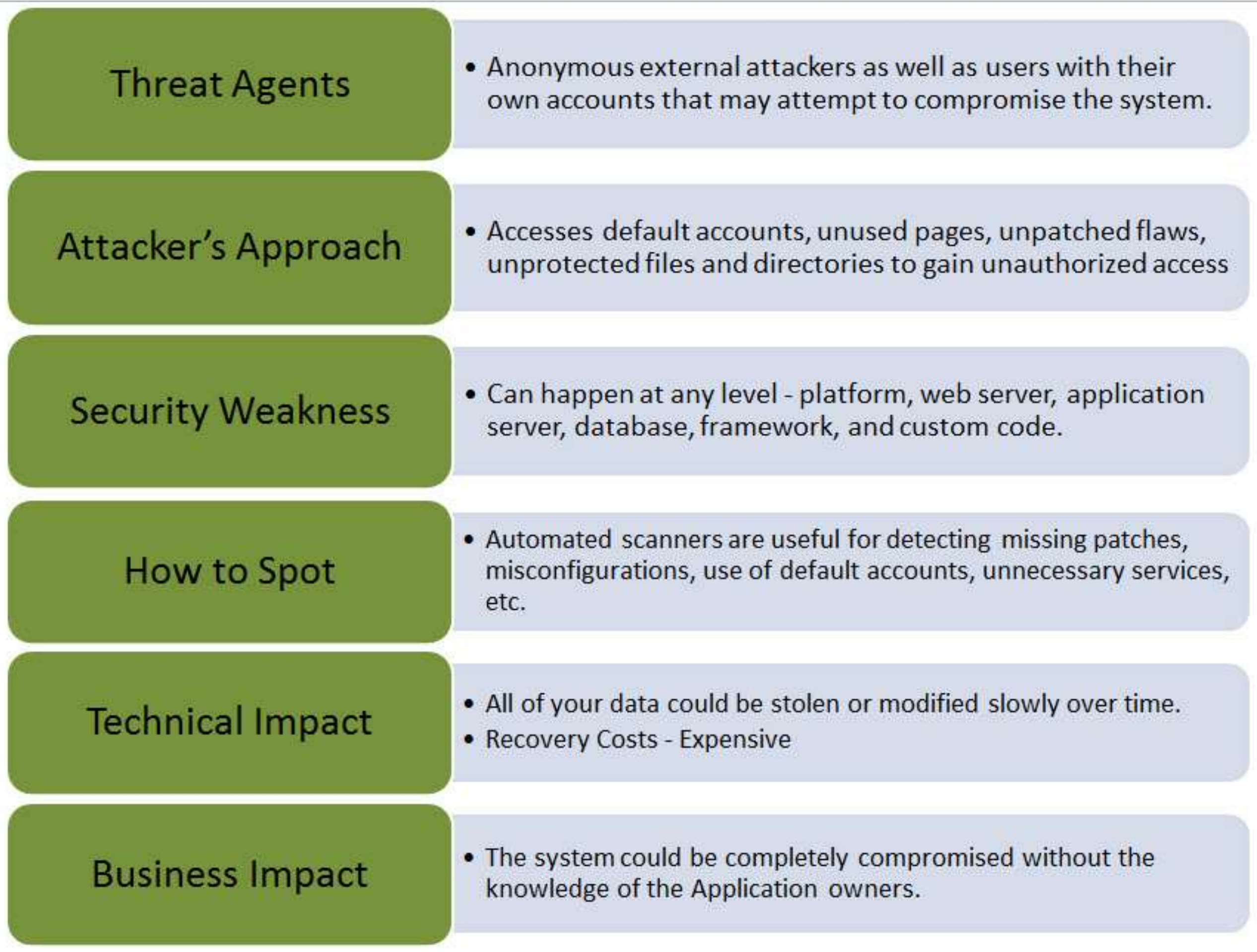
Preventive Mechanisms

Developers can use the following resources/points as a guide to prevent insecure direct object reference during development phase itself.

- Developers should use only one user or session for indirect object references.
- It is also recommended to check the access before using a direct object reference from an untrusted source.

15. Security Misconfiguration

Security Misconfiguration arises when Security settings are defined, implemented, and maintained as defaults. Good security requires a secure configuration defined and deployed for the application, web server, database server, and platform. It is equally important to have the software up to date.



Threat Agents	• Anonymous external attackers as well as users with their own accounts that may attempt to compromise the system.
Attacker's Approach	• Accesses default accounts, unused pages, unpatched flaws, unprotected files and directories to gain unauthorized access
Security Weakness	• Can happen at any level - platform, web server, application server, database, framework, and custom code.
How to Spot	• Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.
Technical Impact	• All of your data could be stolen or modified slowly over time. • Recovery Costs - Expensive
Business Impact	• The system could be completely compromised without the knowledge of the Application owners.

Example

Some classic examples of security misconfiguration are as given:

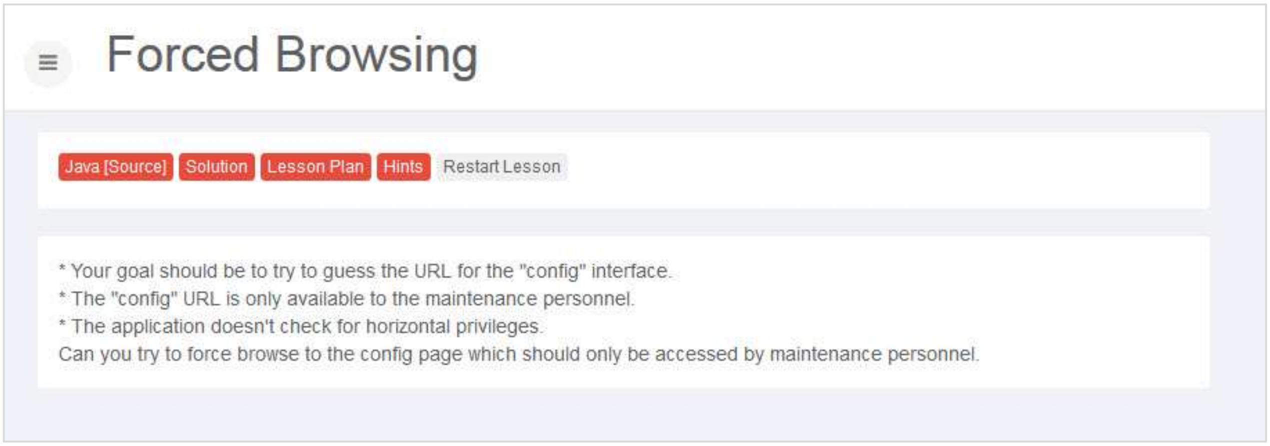
- If Directory listing is not disabled on the server and if attacker discovers the same then the attacker can simply list directories to find any file and execute it. It is also possible to get the actual code base which contains all your custom code and then to find a serious flaws in the application.

- App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers grab those extra information that the error messages provide which is enough for them to penetrate.

- App servers usually come with sample apps that are not well secured. If not removed from production server would result in compromising your server.

Hands ON

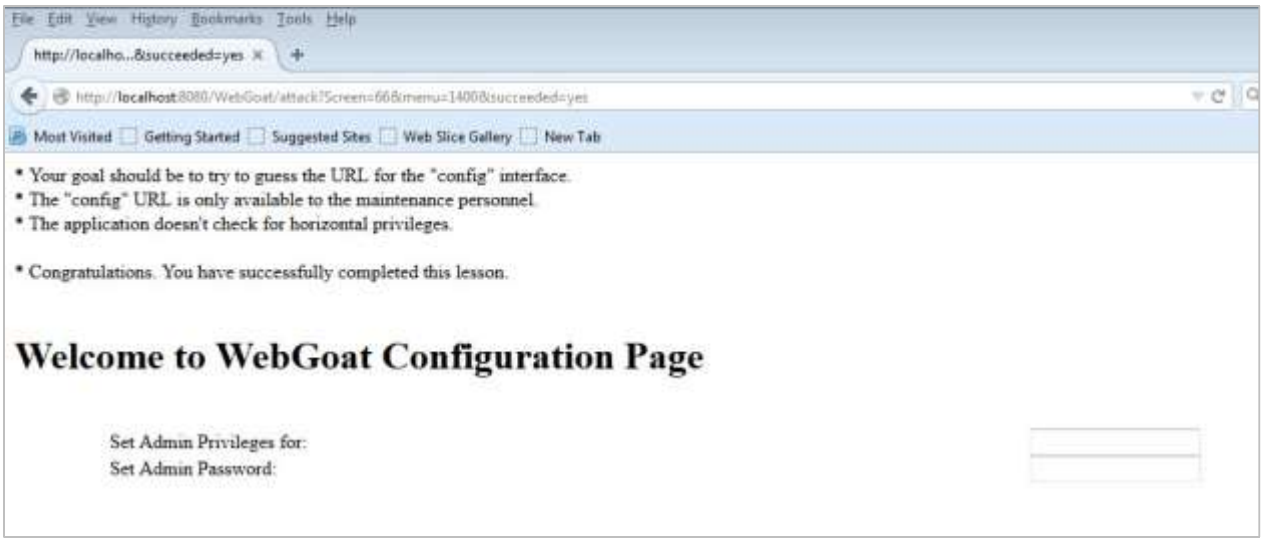
1. Launch Webgoat and navigate to insecure configuration section and let us try to solve that challenge. Snapshot of the same is provided below:



* Your goal should be to try to guess the URL for the "config" interface.
* The "config" URL is only available to the maintenance personnel.
* The application doesn't check for horizontal privileges.
Can you try to force browse to the config page which should only be accessed by maintenance personnel.

2. We can try out as many options as we can think of. All we need to find the URL of config file and we know that the developers follow kind of naming convention for config files. It can be anything that is listed below. It is usually done by BRUTE force technique.
   - web.config
   - config
   - appname.config
   - conf

3. Upon trying various options, we find that 'http://localhost:8080/WebGoat/conf' is successful. The following page is displayed if the attempt is successful:


File Edit View History Bookmarks Tools Help
http://localho...&succeeded=yes
http://localhost:8080/WebGoat/attack?Screen=66&menu=1400&succeeded=yes
Most Visited
Getting Started
Suggested Sites
Web Slice Gallery
New Tab

* Your goal should be to try to guess the URL for the "config" interface.
* The "config" URL is only available to the maintenance personnel.
* The application doesn't check for horizontal privileges.

* Congratulations. You have successfully completed this lesson.

Welcome to WebGoat Configuration Page

Set Admin Privileges for:
Set Admin Password:

Preventive Mechanisms

- All environments such Development, QA, and production environments should be configured identically using different passwords used in each environment that cannot be hacked easily.

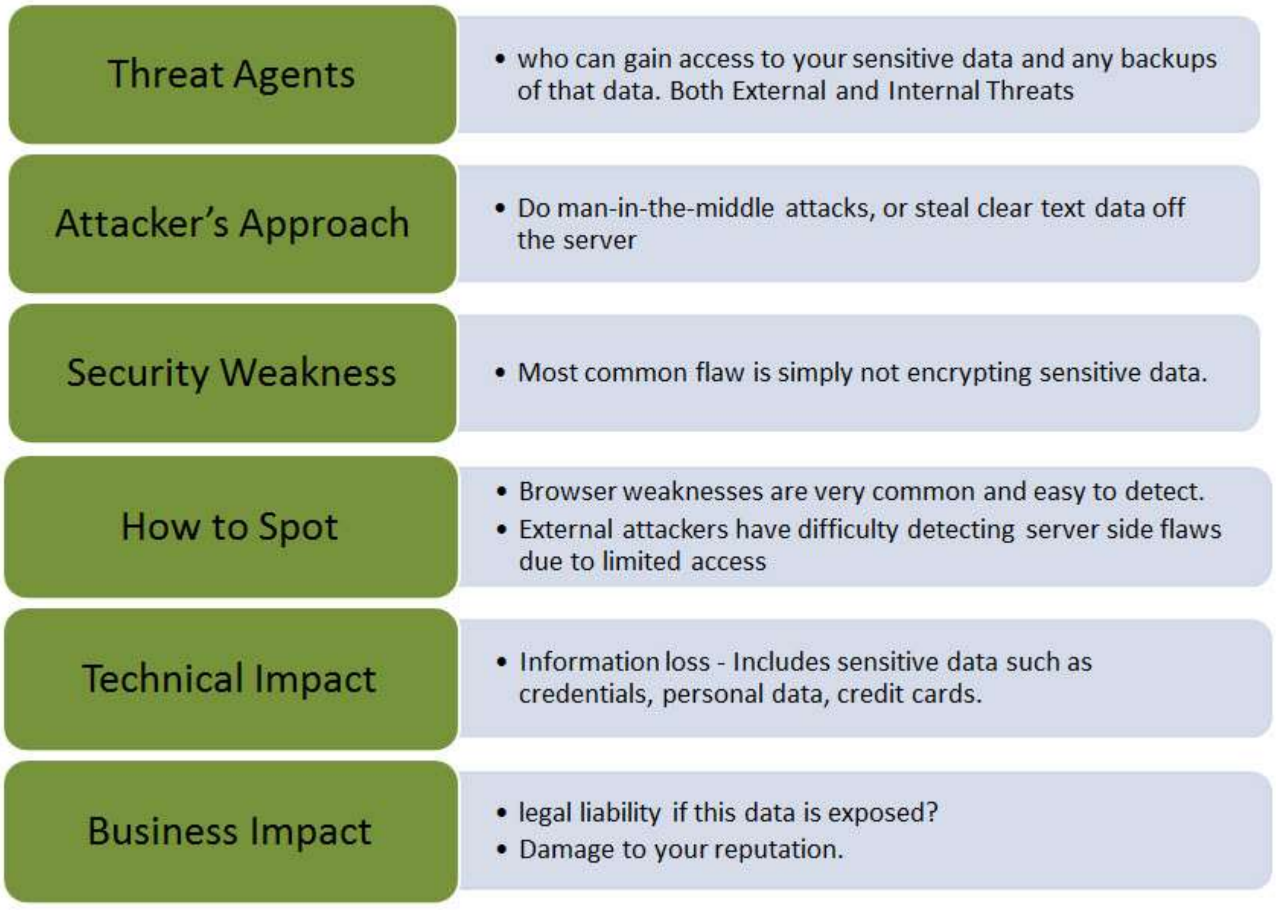
- Ensure that a strong application architecture is being adopted that provides effective, secure separation between components.

- It can also minimize the possibility of this attack by running automated scans and doing audits periodically.

16. Testing Sensitive Data Exposure

As the online applications keep flooding the internet in day by day, not all applications are secured. Many web applications do not properly protect sensitive user data such as credit cards information/Bank account info/authentication credentials. Hackers might end up stealing those weakly protected data to conduct credit card fraud, identity theft, or other crimes.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• who can gain access to your sensitive data and any backups of that data. Both External and Internal Threats
Attacker's Approach	• Do man-in-the-middle attacks, or steal clear text data off the server
Security Weakness	• Most common flaw is simply not encrypting sensitive data.
How to Spot	• Browser weaknesses are very common and easy to detect. • External attackers have difficulty detecting server side flaws due to limited access
Technical Impact	• Information loss - Includes sensitive data such as credentials, personal data, credit cards.
Business Impact	• legal liability if this data is exposed? • Damage to your reputation.

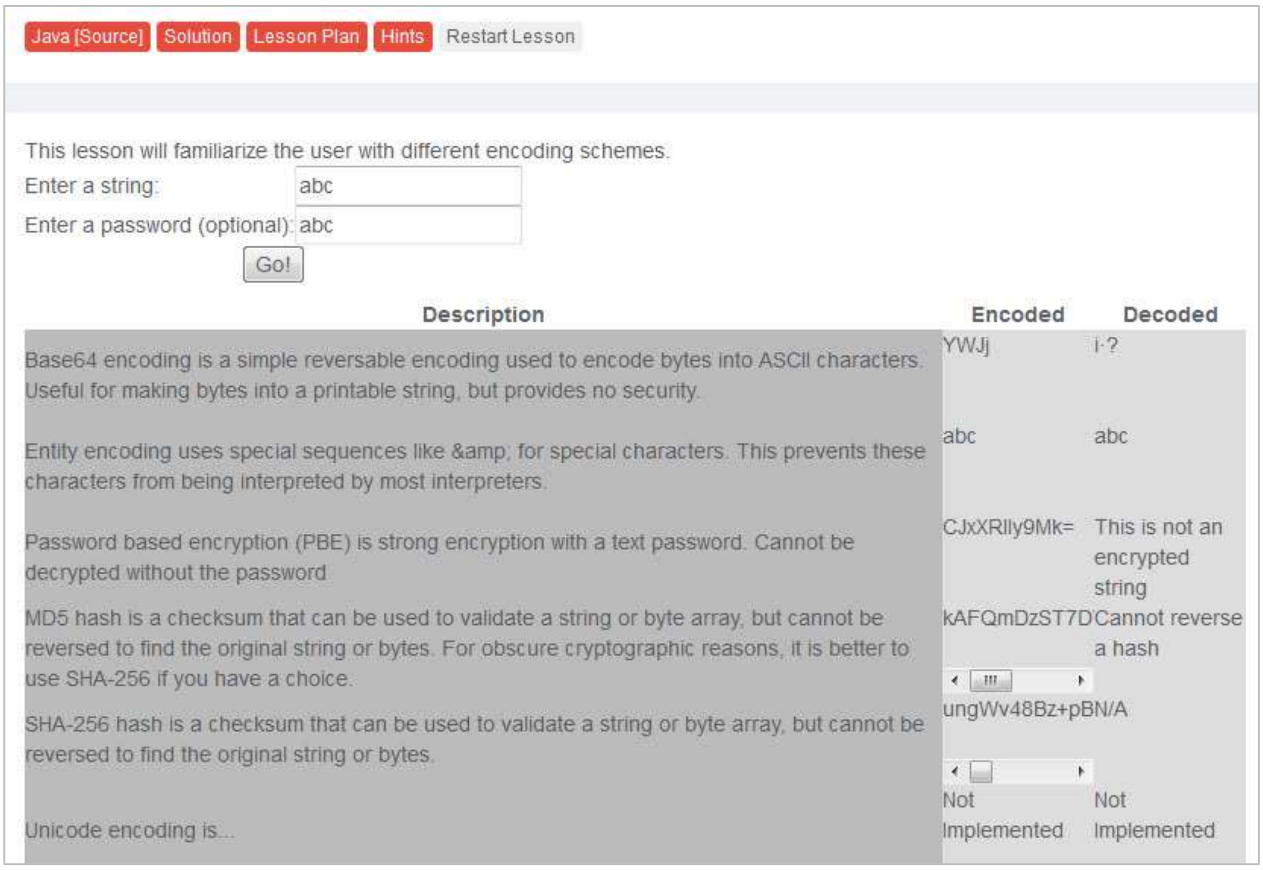
Example

Some of the classic examples of security misconfiguration are as given:

- A site simply does not use SSL for all authenticated pages. This enables an attacker to monitor network traffic and steal the user's session cookie to hijack the users session or accessing their private data.
- An application stores the credit card numbers in an encrypted format in a database. Upon retrieval they are decrypted allowing the hacker to perform a SQL injection attack to retrieve all sensitive info in a clear text. This can be avoided by encrypting the credit card numbers using a public key and allowed back-end applications to decrypt them with the private key.

Hands ON

1. Launch WebGoat and navigate to "Insecure Storage" Section. Snapshot of the same is displayed below.



This lesson will familiarize the user with different encoding schemes.

Enter a string:	abc
Enter a password (optional):	abc

Description	Encoded	Decoded
Base64 encoding is a simple reversable encoding used to encode bytes into ASCII characters. Useful for making bytes into a printable string, but provides no security.	YWJj	i·?
Entity encoding uses special sequences like &amp; for special characters. This prevents these characters from being interpreted by most interpreters.	abc	abc
Password based encryption (PBE) is strong encryption with a text password. Cannot be decrypted without the password	CJxXRIly9Mk=	This is not an encrypted string
MD5 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes. For obscure cryptographic reasons, it is better to use SHA-256 if you have a choice.	kAFQmDzST7D	Cannot reverse a hash
SHA-256 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes.	ungWv48Bz+pB	N/A
Unicode encoding is...	Not Implemented	Not Implemented

2. Enter the username and password. It is time to learn different kind of encoding and encryption methodologies that we discussed previously.

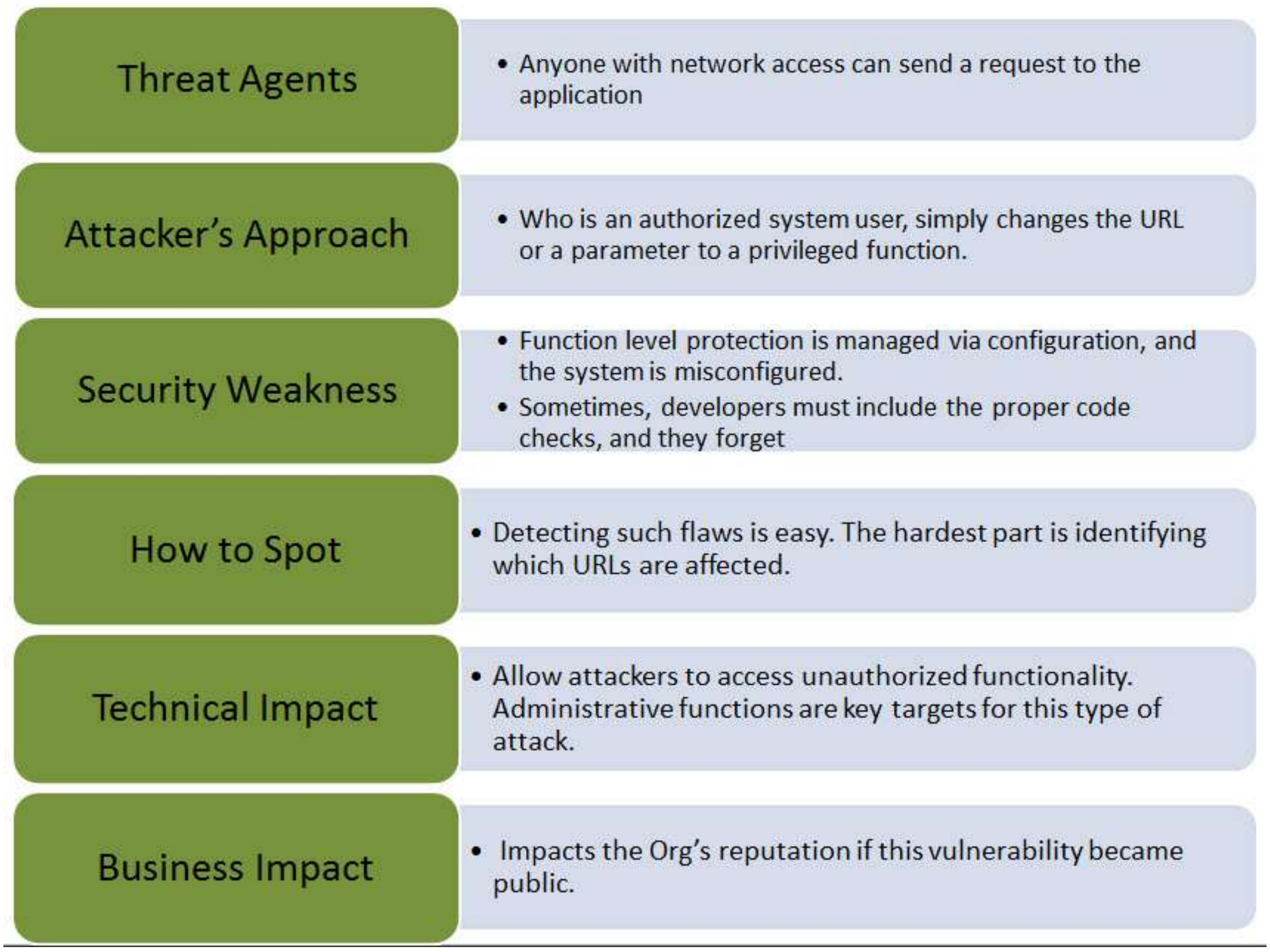
Preventive Mechanisms

- It is advised not to store sensitive data unnecessarily and should be scraped as soon as possible if it is no more required.
- It is important to ensure that we incorporate strong and standard encryption algorithms are used and proper key management is in place.
- It can also be avoided by disabling autocomplete on forms that collect sensitive data such as password and disable caching for pages that contain sensitive data.

17. Missing Function Level Access Control

Most of the web applications verify function level access rights before making that functionality accessible to the user. However, if the same access control checks are not performed on the server, hackers are able to penetrate into the application without proper authorization.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Anyone with network access can send a request to the application
Attacker's Approach	• Who is an authorized system user, simply changes the URL or a parameter to a privileged function.
Security Weakness	• Function level protection is managed via configuration, and the system is misconfigured. • Sometimes, developers must include the proper code checks, and they forget
How to Spot	• Detecting such flaws is easy. The hardest part is identifying which URLs are affected.
Technical Impact	• Allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack.
Business Impact	• Impacts the Org's reputation if this vulnerability became public.

Example

Here is a classic example of Missing Function Level Access Control:

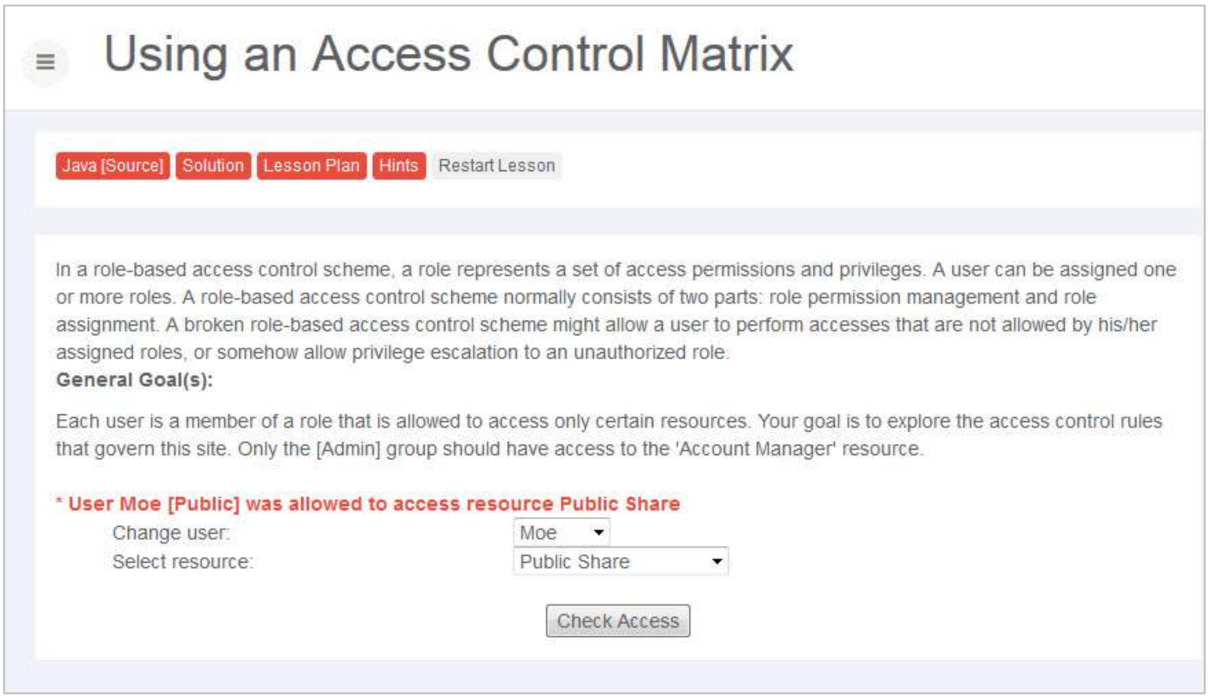
The hacker simply forces target URLs. Usually admin access requires authentication, however, if the application access is not verified, then an unauthenticated user can access admin page.

```
' Below URL might be accessible to an authenticated user
http://website.com/app/standarduserpage
```

```
' A NON Admin user is able to access admin page without authorization.
http://website.com/app/admin_page
```

Hands ON

1. Let us login as account manager by first going through the list of users and their access privileges.



Using an Access Control Matrix

In a role-based access control scheme, a role represents a set of access permissions and privileges. A user can be assigned one or more roles. A role-based access control scheme normally consists of two parts: role permission management and role assignment. A broken role-based access control scheme might allow a user to perform accesses that are not allowed by his/her assigned roles, or somehow allow privilege escalation to an unauthorized role.

General Goal(s):

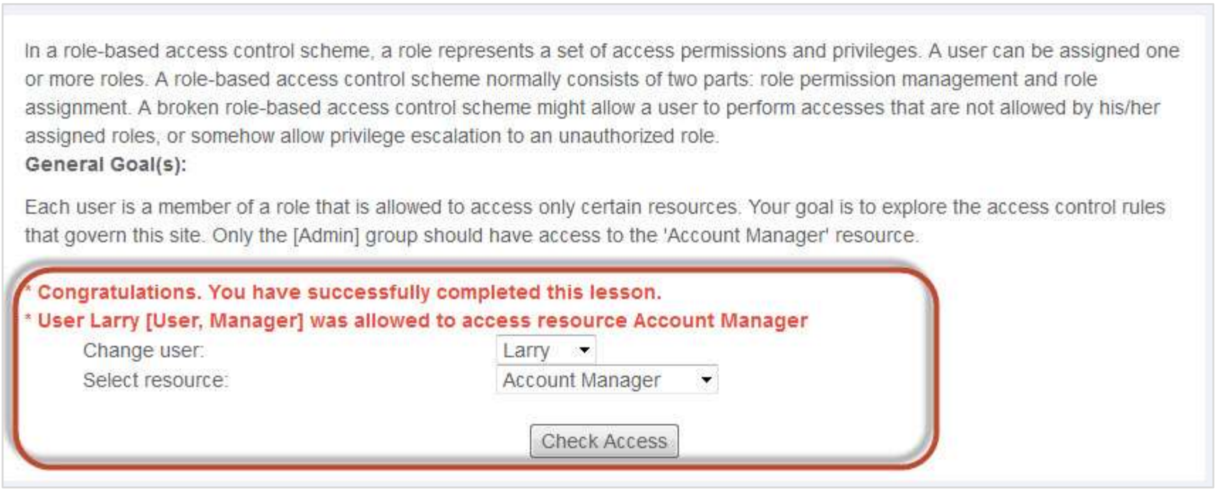
Each user is a member of a role that is allowed to access only certain resources. Your goal is to explore the access control rules that govern this site. Only the [Admin] group should have access to the 'Account Manager' resource.

* User Moe [Public] was allowed to access resource Public Share

Change user: Moe

Select resource: Public Share

2. Upon trying various combinations we can find out that Larry has access to resource account manager.



In a role-based access control scheme, a role represents a set of access permissions and privileges. A user can be assigned one or more roles. A role-based access control scheme normally consists of two parts: role permission management and role assignment. A broken role-based access control scheme might allow a user to perform accesses that are not allowed by his/her assigned roles, or somehow allow privilege escalation to an unauthorized role.

General Goal(s):

Each user is a member of a role that is allowed to access only certain resources. Your goal is to explore the access control rules that govern this site. Only the [Admin] group should have access to the 'Account Manager' resource.

* Congratulations. You have successfully completed this lesson.

* User Larry [User, Manager] was allowed to access resource Account Manager

Change user: Larry

Select resource: Account Manager

Preventive Mechanisms

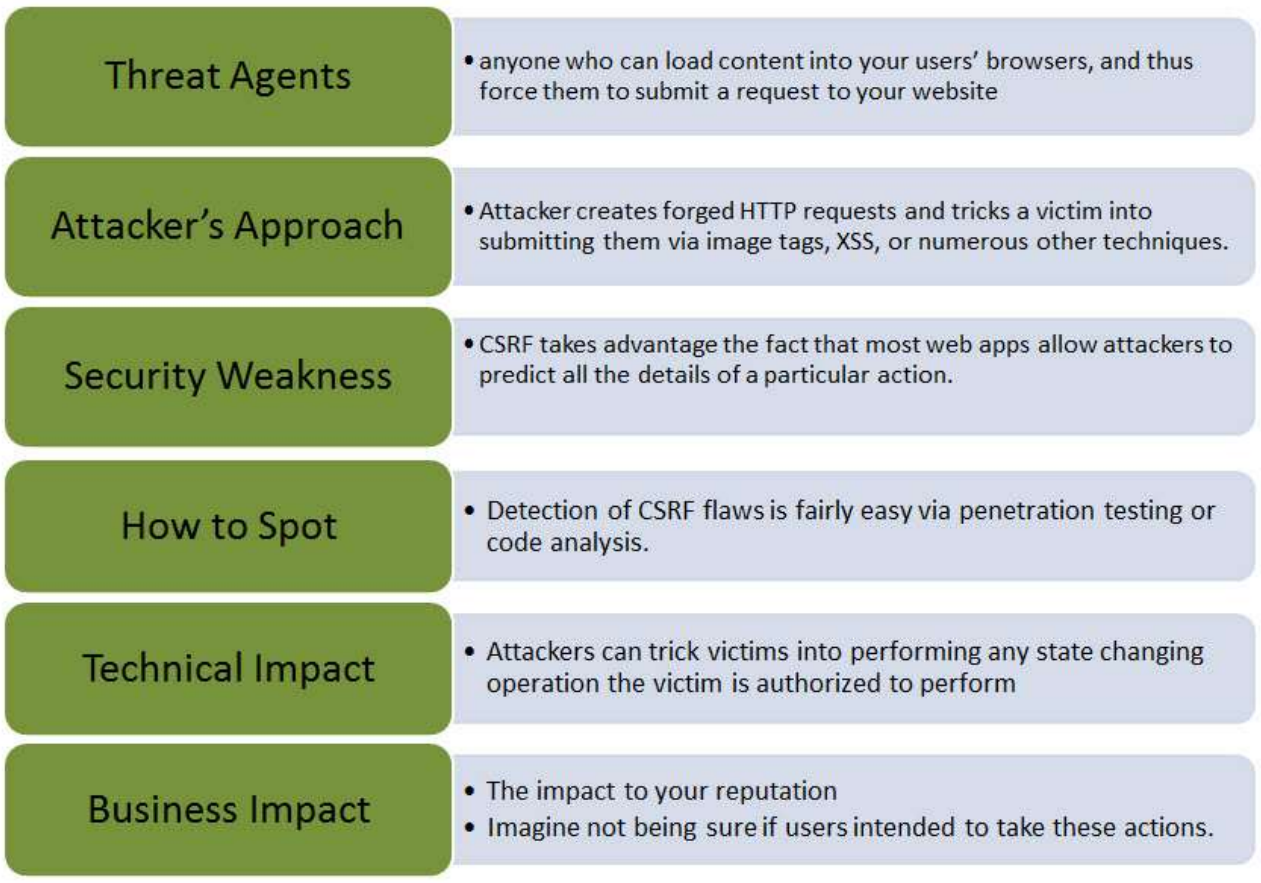
- The authentication mechanism should deny all access by default, and provide access to specific roles for every function.

- In a workflow based application, verify the users’ state before allowing them to access any resources.

18. Cross-Site Request Forgery (CSRF)

A CSRF attack forces an authenticated user (victim) to send a forged HTTP request, including the victim's session cookie to a vulnerable web application, which allows the attacker to force the victim's browser to generate request such that the vulnerable app perceives as legitimate requests from the victim.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• anyone who can load content into your users' browsers, and thus force them to submit a request to your website
Attacker's Approach	• Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques.
Security Weakness	• CSRF takes advantage the fact that most web apps allow attackers to predict all the details of a particular action.
How to Spot	• Detection of CSRF flaws is fairly easy via penetration testing or code analysis.
Technical Impact	• Attackers can trick victims into performing any state changing operation the victim is authorized to perform
Business Impact	• The impact to your reputation • Imagine not being sure if users intended to take these actions.

Example

Here is a classic example of CSRF:

1. Let us say, the vulnerable application sends a state changing request as a plain text without any encryption.

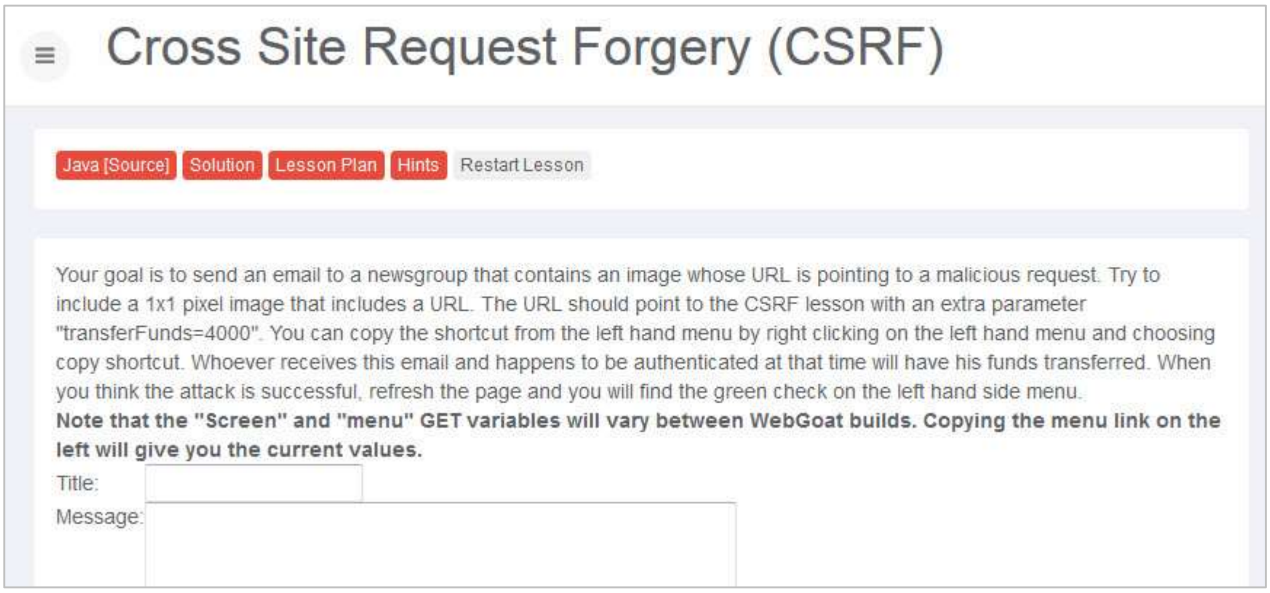
```
http://bankx.com/app?action=transferFund&amount=3500&destinationAccount=4673243243
```

2. Now the hacker constructs a request that transfers money from the victim's account to the attacker's account by embedding the request in an image that is stored on various sites under the attacker's control:

```
<img
src="http://bankx.com/app?action=transferFunds&amount=14000&destinationAccount=atta
ckersAcct#" width="0" height="0" />
```

Hands ON

1. Let us perform a CSRF forgery by embedding a Java script into an image. The snapshot of the problem is listed below.

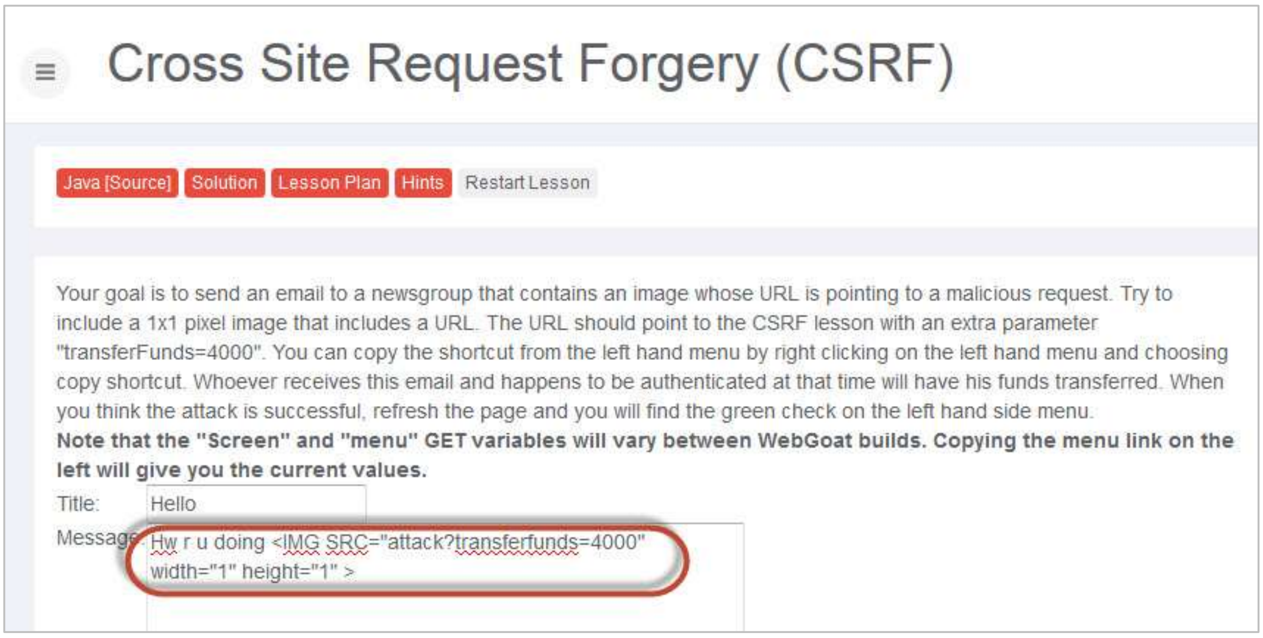


Cross Site Request Forgery (CSRF)

Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.
Note that the "Screen" and "menu" GET variables will vary between WebGoat builds. Copying the menu link on the left will give you the current values.

Title:
Message:

2. Now we need to mock up the transfer into a 1x1 image and make the victim to click on the same.

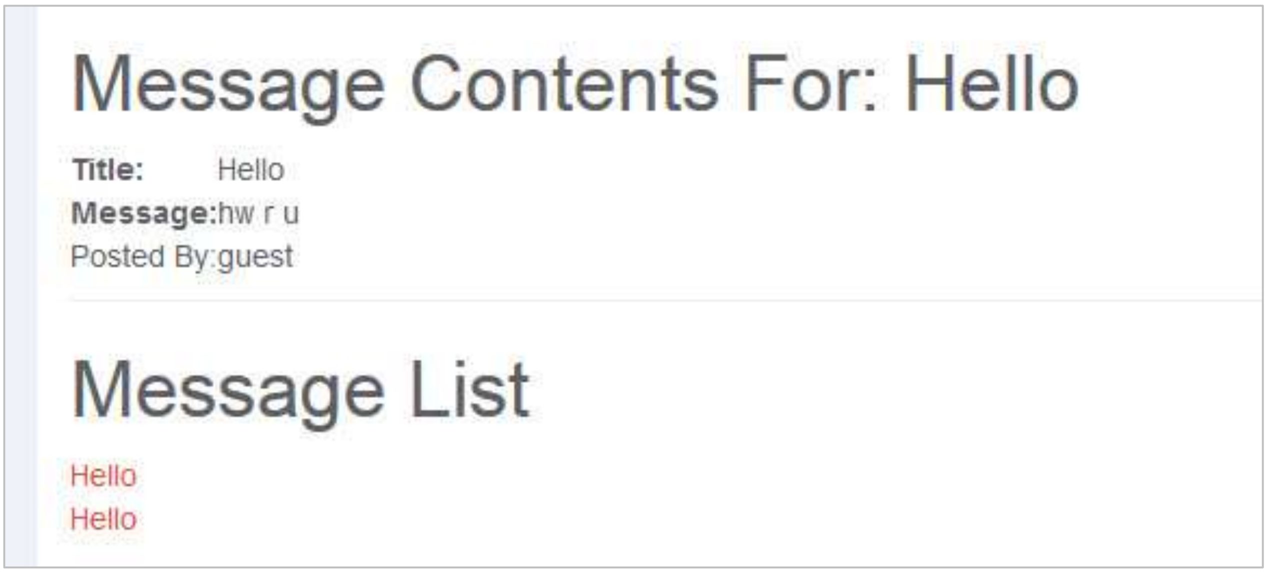


Cross Site Request Forgery (CSRF)

Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.
Note that the "Screen" and "menu" GET variables will vary between WebGoat builds. Copying the menu link on the left will give you the current values.

Title: Hello
Message: Hw r u doing <IMG SRC="attack?transferfunds=4000" width="1" height="1" >

3. Upon submitting the message, the message is displayed as highlighted below.



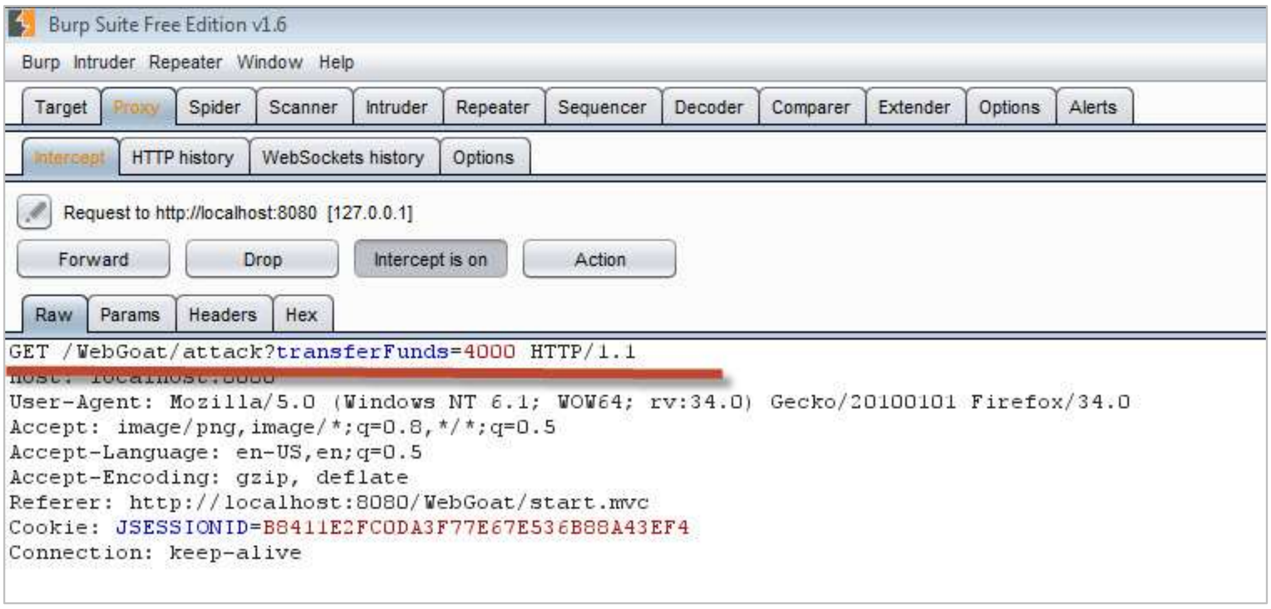
Message Contents For: Hello

Title: Hello
Message:hw r u
Posted By:guest

Message List

Hello
Hello

4. Now if the victim clicks the following URL, the transfer is executed, which can be found intercepting the user action using burp suite. We are able to see the transfer by spotting it in Get message as shown below:


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
GET /WebGoat/attack?transferFunds=4000 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=B8411E2FC0DA3F77E67E536B88A43EF4
Connection: keep-alive
```

5. Now upon clicking refresh, the lesson completion mark is shown.

Preventive Mechanisms

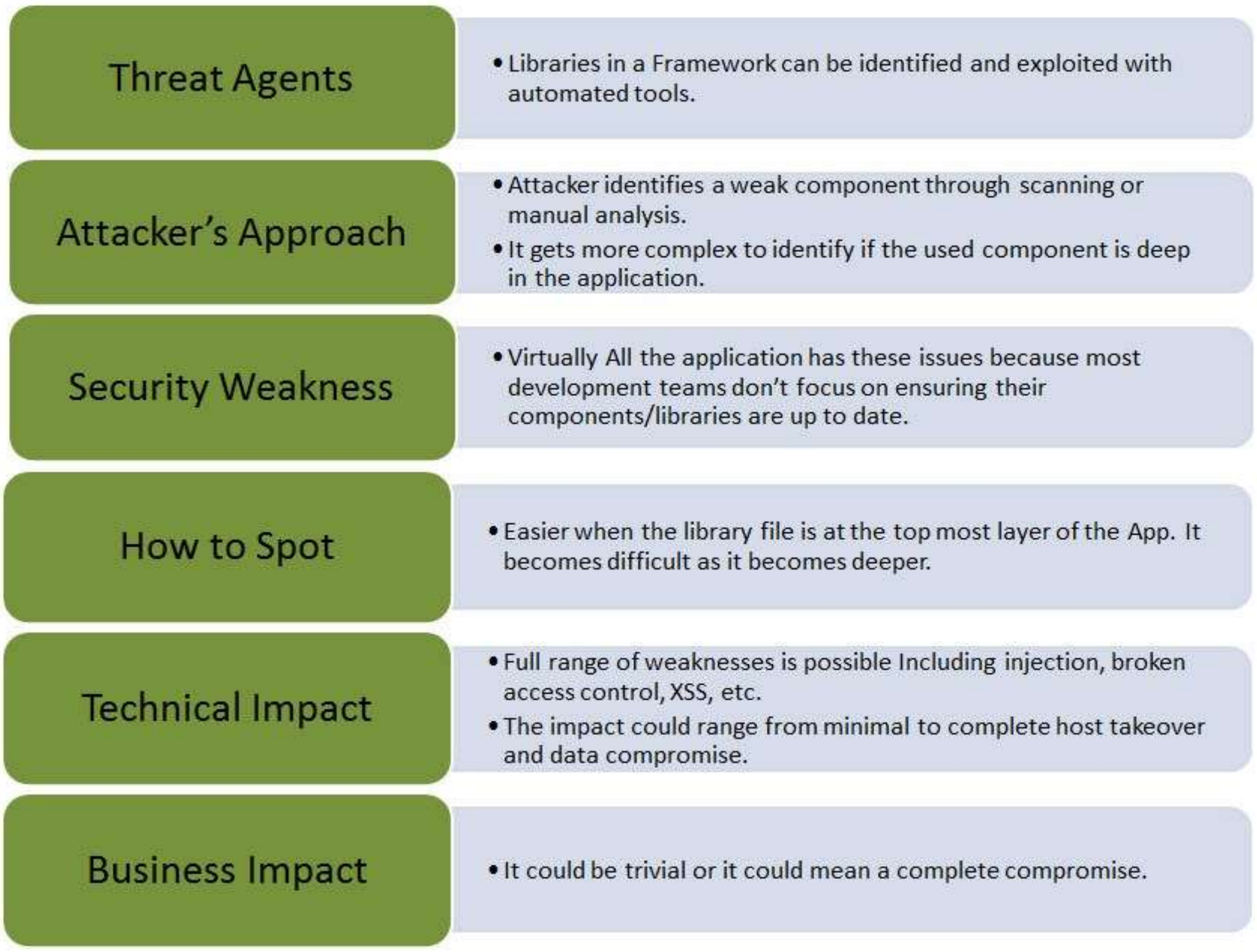
- CSRF can be avoided by creating a unique token in a hidden field which would be sent in the body of the HTTP request rather than in an URL, which is more prone to exposure.

  Forcing the user to re-authenticate or proving that they are users in order to protect CSRF. For example, CAPTCHA.

19. Components with Vulnerabilities

This kind of threat occurs when the components such as libraries and frameworks used within the app almost always execute with full privileges. If a vulnerable component is exploited, it makes the hacker’s job easier to cause a serious data loss or server takeover.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



Threat Agents	• Libraries in a Framework can be identified and exploited with automated tools.
Attacker’s Approach	• Attacker identifies a weak component through scanning or manual analysis. • It gets more complex to identify if the used component is deep in the application.
Security Weakness	• Virtually All the application has these issues because most development teams don’t focus on ensuring their components/libraries are up to date.
How to Spot	• Easier when the library file is at the top most layer of the App. It becomes difficult as it becomes deeper.
Technical Impact	• Full range of weaknesses is possible Including injection, broken access control, XSS, etc. • The impact could range from minimal to complete host takeover and data compromise.
Business Impact	• It could be trivial or it could mean a complete compromise.

Example

The following examples are of using components with known vulnerabilities:

- Attackers can invoke any web service with full permission by failing to provide an identity token.

- Remote-code execution with Expression Language injection vulnerability is introduced through the Spring Framework for Java based apps.

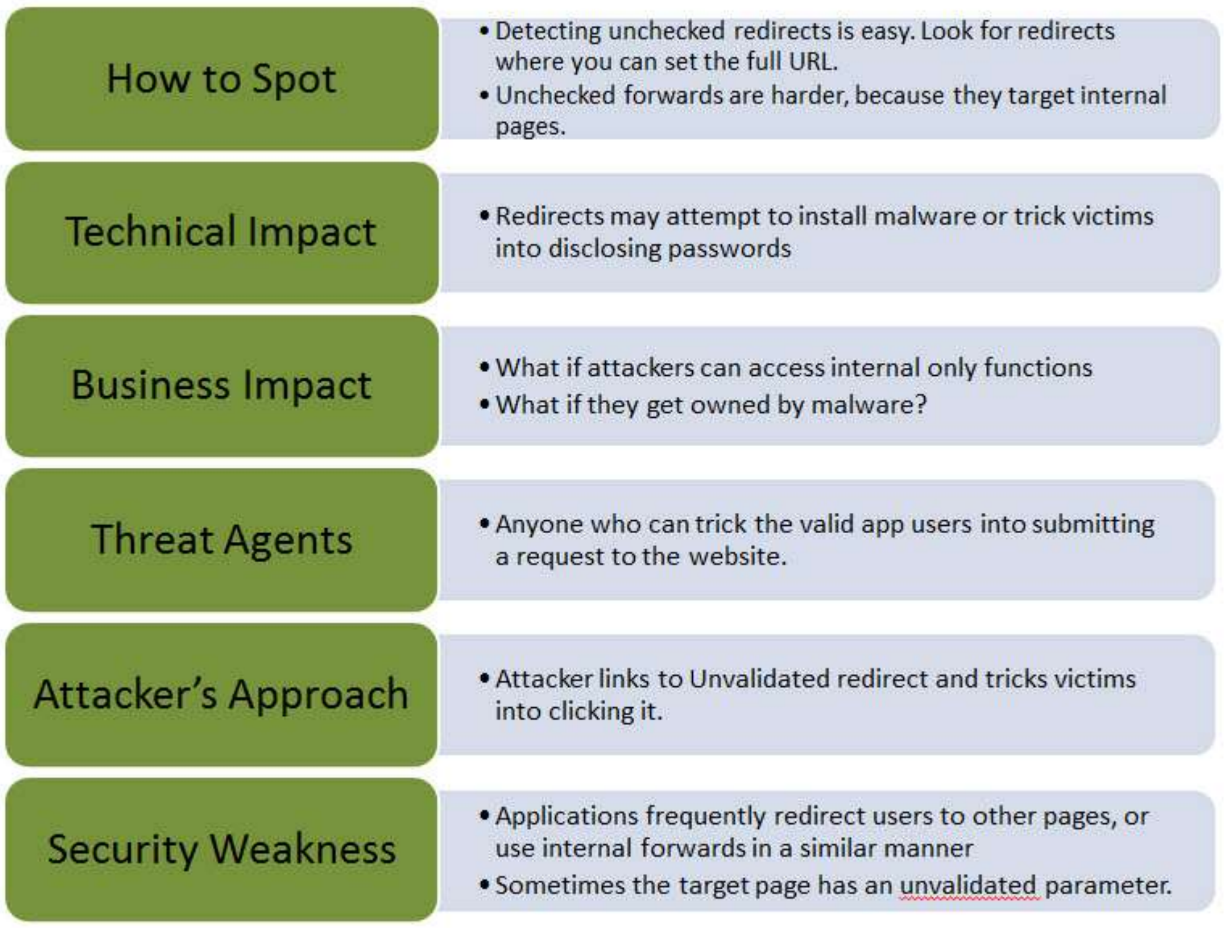
Preventive Mechanisms

- Identify all components and the versions that are being used in the webapps not just restricted to database/frameworks.
- Keep all the components such as public databases, project mailing lists etc. up to date.
- Add security wrappers around components that are vulnerable in nature.

20. Unvalidated Redirects and Forwards

Most web applications on the internet frequently redirect and forward users to other pages or other external websites. However, without validating the credibility of those pages, hackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.



How to Spot	• Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. • Unchecked forwards are harder, because they target internal pages.
Technical Impact	• Redirects may attempt to install malware or trick victims into disclosing passwords
Business Impact	• What if attackers can access internal only functions • What if they get owned by malware?
Threat Agents	• Anyone who can trick the valid app users into submitting a request to the website.
Attacker's Approach	• Attacker links to Unvalidated redirect and tricks victims into clicking it.
Security Weakness	• Applications frequently redirect users to other pages, or use internal forwards in a similar manner • Sometimes the target page has an unvalidated parameter.

Example

Some classic examples of Unvalidated Redirects and Forwards are as given:

1. Let us say the application has a page - redirect.jsp, which takes a parameter *redirectrul*. The hacker adds a malicious URL that redirects users which performs phishing/installs malware.

```
http://www.mywebapp.com/redirect.jsp?redirectrul=hacker.com
```

2. All web application used to forward users to different parts of the site. In order to achieve the same, some pages use a parameter to indicate where the user should be redirected if an operation is successful. The attacker crafts an URL that passes the application's access control check and then forwards the attacker to administrative functionality for which the attacker has not got the access.

```
http://www.mywebapp.com/checkstatus.jsp?fwd=appadmin.jsp
```

Preventive Mechanisms

- It is better to avoid using redirects and forwards.
- If it is unavoidable, then it should be done without involving user parameters in redirecting the destination.

21. Ajax Security

Asynchronous Javascript and XML (AJAX) is one of the latest techniques used to develope web application inorder to give a rich user experience. Since it is a new technology, there are many security issues that are yet to be completed established and below are the few security issues in AJAX.

- The attack surface is more as there are more inputs to be secured.
- It also exposes the internal functions of the applications.
- Failure to protect authentication information and sessions.
- There is a very narrow line between client-side and server-side, hence there are possibilities of committing security mistakes.

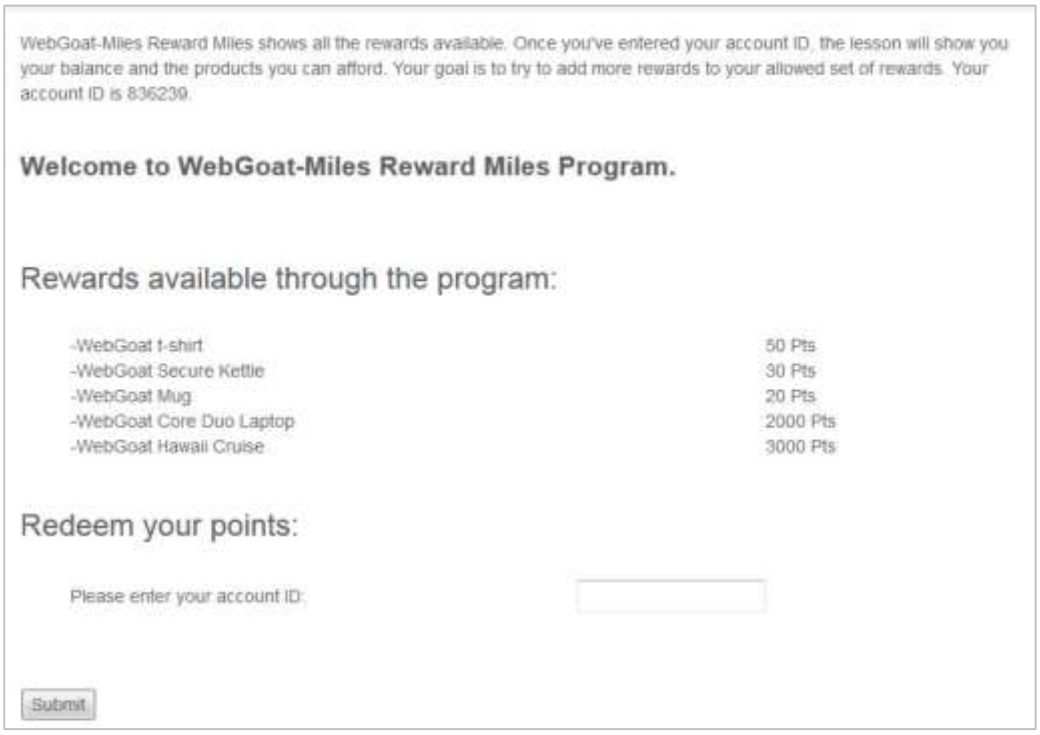
Example

Here is an example for AJAX Security:

In 2006, a worm infected yahoo mail service using XSS and AJAX that took advantage of a vulnerability in Yahoo Mail's *onload* event handling. When an infected email was opened, the worm executed its JavaScript, sending a copy to all the Yahoo contacts of the infected user.

Hands ON

1. We need to try to add more rewards to your allowed set of reward using XML injection. Below is the snapshot of the scenario.



WebGoat-Miles Reward Miles shows all the rewards available. Once you've entered your account ID, the lesson will show you your balance and the products you can afford. Your goal is to try to add more rewards to your allowed set of rewards. Your account ID is 836239.

Welcome to WebGoat-Miles Reward Miles Program.

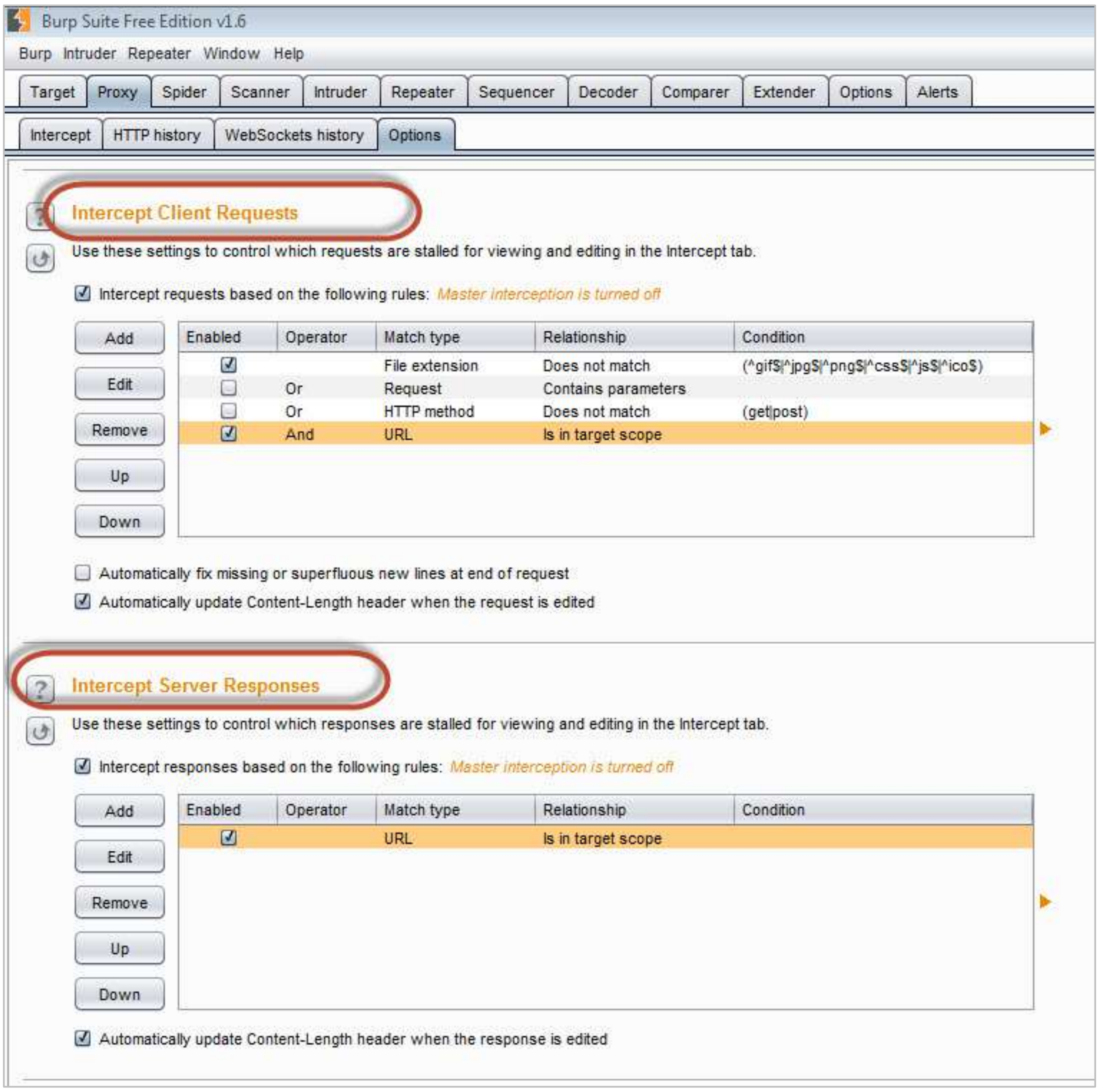
Rewards available through the program:

-WebGoat t-shirt	50 Pts
-WebGoat Secure Kettle	30 Pts
-WebGoat Mug	20 Pts
-WebGoat Core Duo Laptop	2000 Pts
-WebGoat Hawaii Cruise	3000 Pts

Redeem your points:

Please enter your account ID:

2. Make sure that we intercept both request and response using Burp Suite. Settings of the same as shown below.


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Up
Down

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

☑ Intercept requests based on the following rules: Master interception is turned off

Enabled	Operator	Match type	Relationship	Condition
☑		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$\|^ico$)
☐	Or	Request	Contains parameters	
☐	Or	HTTP method	Does not match	(get|post)
☑	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request

☑ Automatically update Content-Length header when the request is edited

Intercept Server Responses

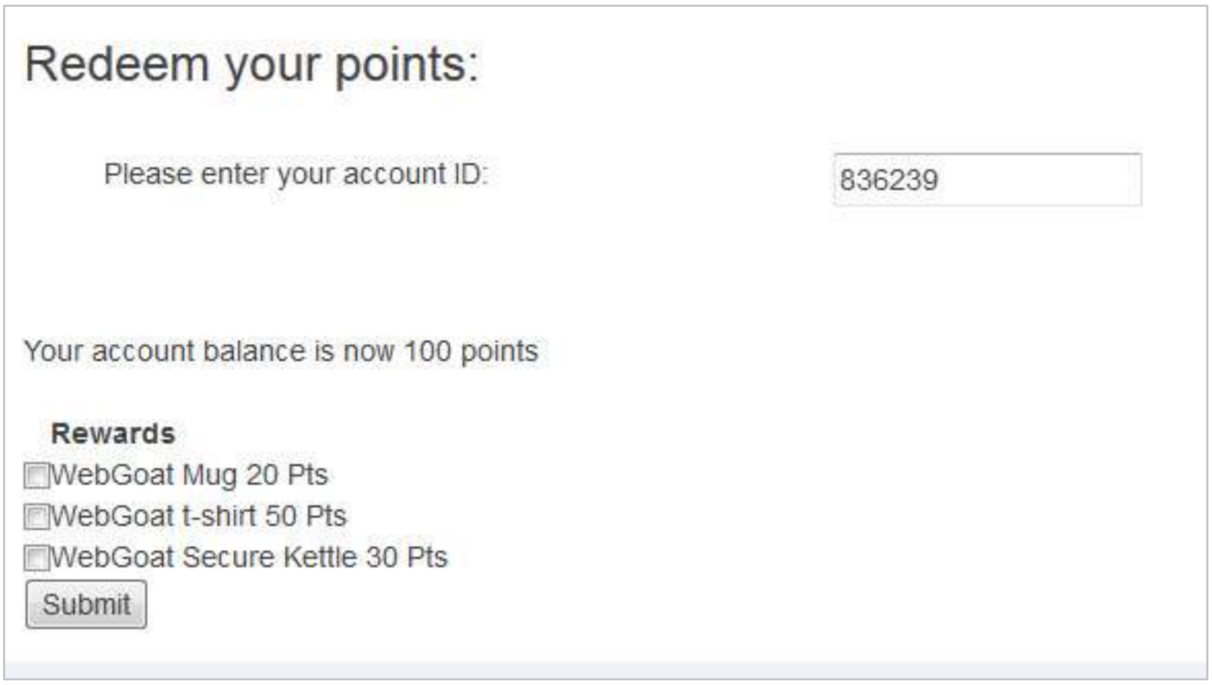
Use these settings to control which responses are stalled for viewing and editing in the Intercept tab.

☑ Intercept responses based on the following rules: Master interception is turned off

Enabled	Operator	Match type	Relationship	Condition
☑		URL	Is in target scope	

☑ Automatically update Content-Length header when the response is edited

3. Enter the account number as given in the scenario. We will be able to get a list of all rewards that we are eligible for. We are eligible for 3 rewards out of 5.


Redeem your points:

Please enter your account ID: 836239

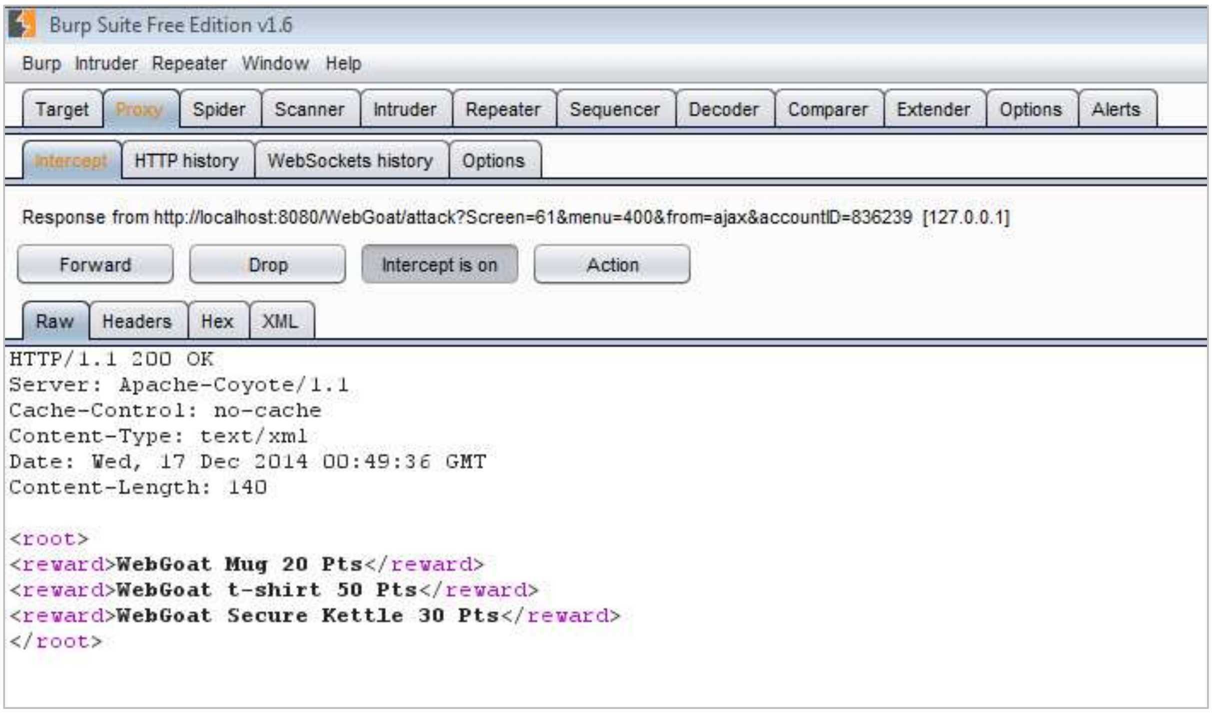
Your account balance is now 100 points

Rewards

- WebGoat Mug 20 Pts
- WebGoat t-shirt 50 Pts
- WebGoat Secure Kettle 30 Pts

Submit

4. Now let us click 'Submit' and see what we get in the response XML. As shown below the three rewards that are we are eligible are passed to us as XML.

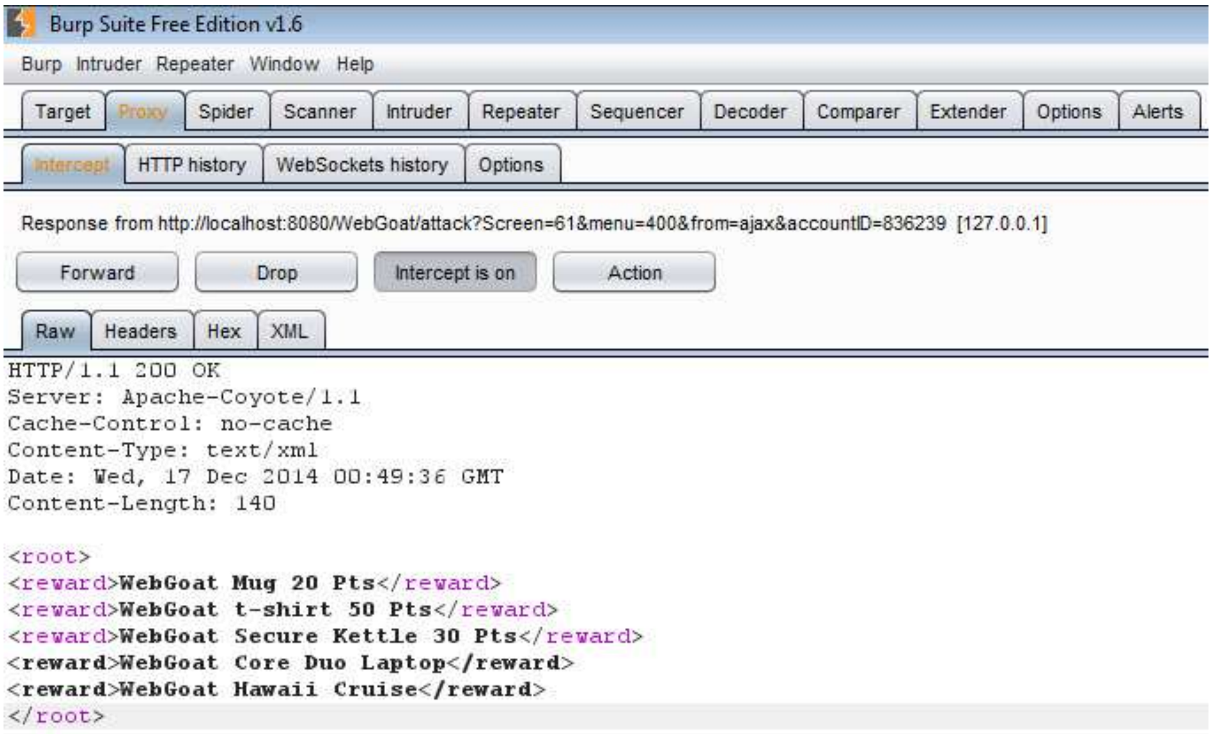

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Headers Hex XML

Response from http://localhost:8080/WebGoat/attack?Screen=61&menu=400&from=ajax&accountID=836239 [127.0.0.1]

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Cache-Control: no-cache
Content-Type: text/xml
Date: Wed, 17 Dec 2014 00:49:36 GMT
Content-Length: 140

<root>
<reward>WebGoat Mug 20 Pts</reward>
<reward>WebGoat t-shirt 50 Pts</reward>
<reward>WebGoat Secure Kettle 30 Pts</reward>
</root>
```

5. Now let us edit those XMLs and add the other two rewards as well.

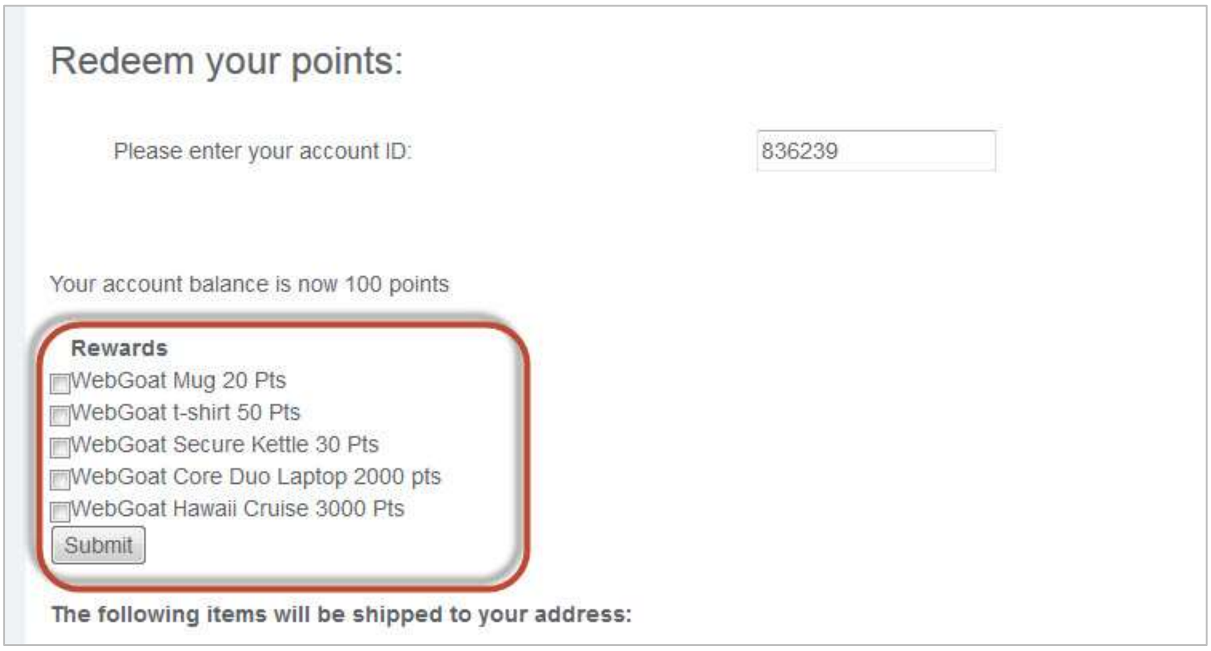

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Headers Hex XML

Response from http://localhost:8080/WebGoat/attack?Screen=61&menu=400&from=ajax&accountID=836239 [127.0.0.1]

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Cache-Control: no-cache
Content-Type: text/xml
Date: Wed, 17 Dec 2014 00:49:36 GMT
Content-Length: 140

<root>
<reward>WebGoat Mug 20 Pts</reward>
<reward>WebGoat t-shirt 50 Pts</reward>
<reward>WebGoat Secure Kettle 30 Pts</reward>
<reward>WebGoat Core Duo Laptop</reward>
<reward>WebGoat Hawaii Cruise</reward>
</root>
```

6. Now all the rewards would be displayed to the user for them to select. Select the ones that we added and click 'Submit'.



Redeem your points:

Please enter your account ID: 836239

Your account balance is now 100 points

Rewards
- [ ] WebGoat Mug 20 Pts
- [ ] WebGoat t-shirt 50 Pts
- [ ] WebGoat Secure Kettle 30 Pts
- [ ] WebGoat Core Duo Laptop 2000 pts
- [ ] WebGoat Hawaii Cruise 3000 Pts

The following items will be shipped to your address:

7. The following message appears saying, "* Congratulations. You have successfully completed this lesson."

Preventive Mechanisms

Client side:

- Use .innerText instead of .innerHtml.
- Do not use eval.
- Do not rely on client logic for security.
- Avoid writing serialization code.
- Avoid building XML dynamically.
- Never transmit secrets to the client.
- Do not perform encryption in client side code.
- Do not perform security impacting logic on client side.

Server side:

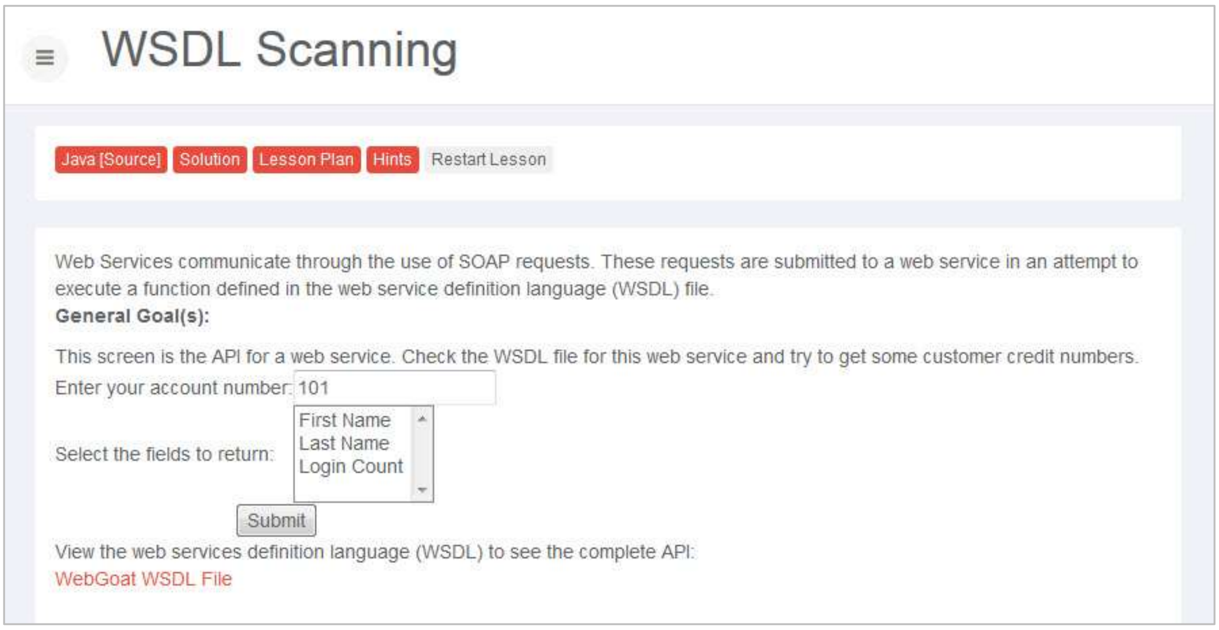
- Use CSRF protection.
- Avoid writing serialization code.
- Services can be called by users directly.
- Avoid building XML by hand, use the framework.
- Avoid building JSON by hand, use an existing framework.

22. Web Service Security

In modern web-based applications, the usage of web services is inevitable and they are prone for attacks as well. Since the web services request fetch from multiple websites developers have to take few additional measures in order to avoid any kind of penetration by hackers.

Hands ON

1. Navigate to web services area of Webgoat and go to WSDL Scanning. We need to now get credit card details of some other account number. Snapshot of the scenario is as mentioned below.



WSDL Scanning

Web Services communicate through the use of SOAP requests. These requests are submitted to a web service in an attempt to execute a function defined in the web service definition language (WSDL) file.

General Goal(s):

This screen is the API for a web service. Check the WSDL file for this web service and try to get some customer credit numbers.

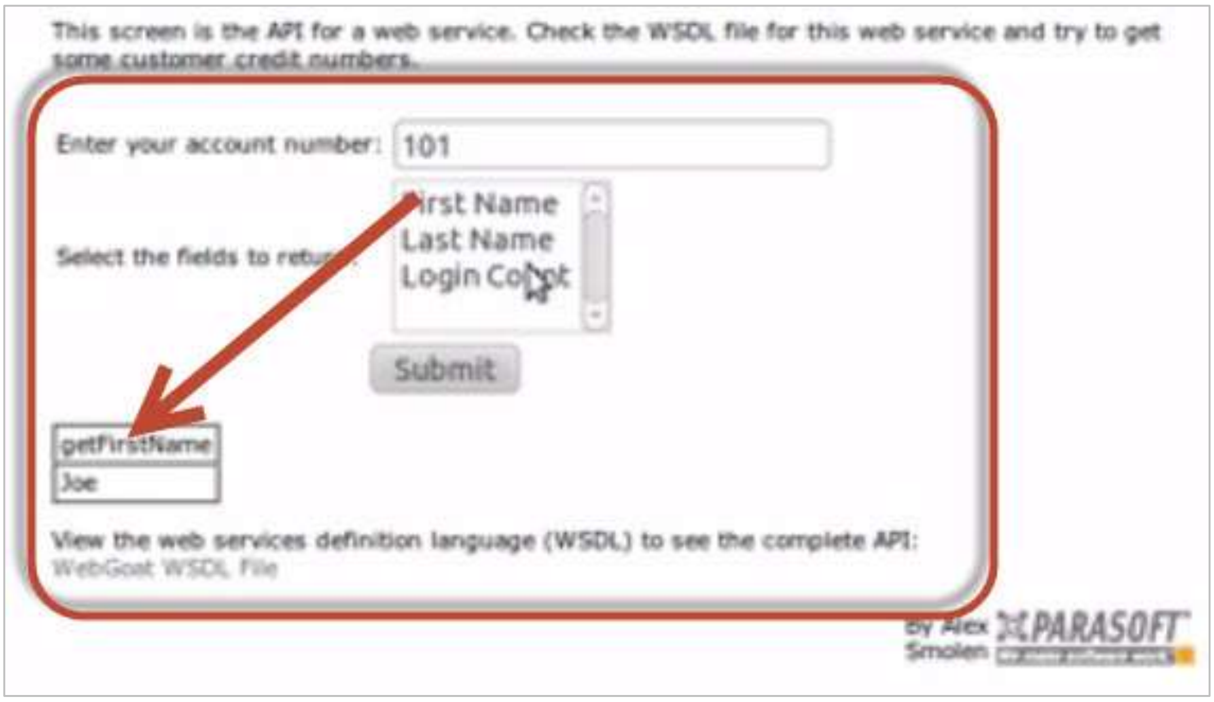
Enter your account number: 101

Select the fields to return: First Name, Last Name, Login Count

View the web services definition language (WSDL) to see the complete API:

WebGoat WSDL File

2. If we select the first name, the 'getFirstName' function call is made through SOAP request xml.



This screen is the API for a web service. Check the WSDL file for this web service and try to get some customer credit numbers.

Enter your account number: 101

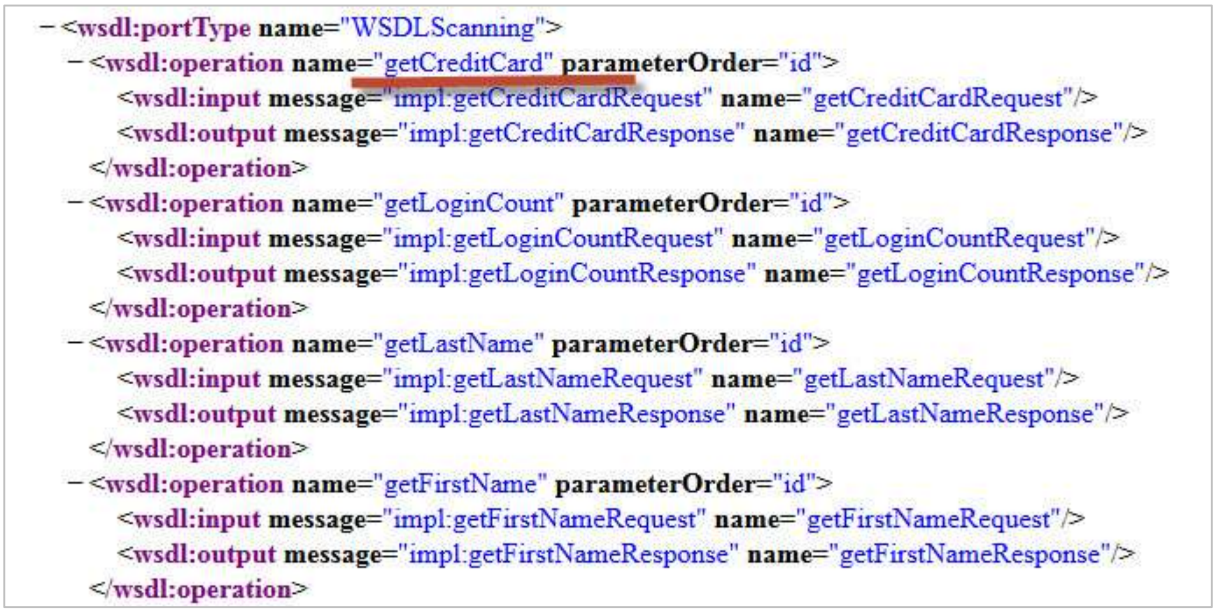
Select the fields to return: First Name, Last Name, Login Count

Submit

getFirstName
Joe

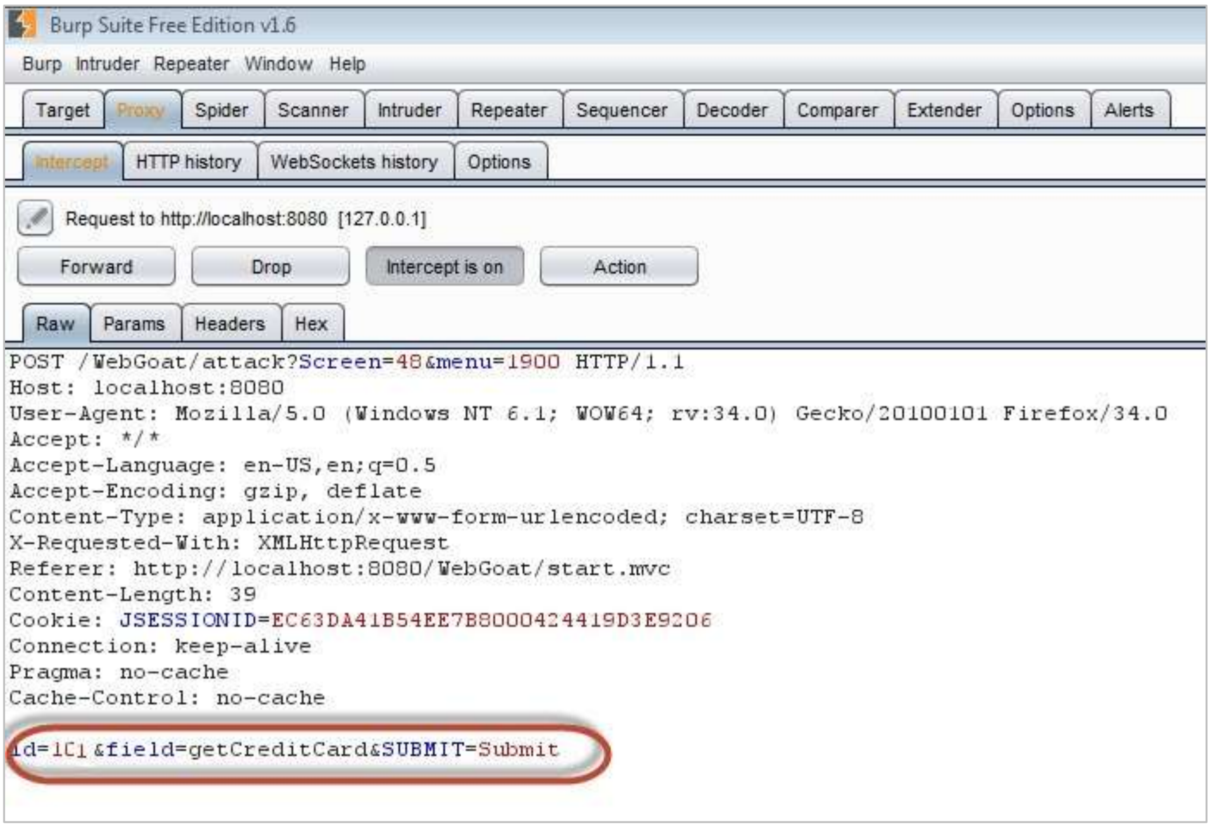
View the web services definition language (WSDL) to see the complete API:
WebGoat WSDL File

3. By opening the WSDL, we are can see that there is a method to retrieve credit card information as well 'getCreditCard'. Now let us tamper the inputs using Burp suite as shown below:



```
- <wsdl:portType name="WSDLScanning">
  - <wsdl:operation name="getCreditCard" parameterOrder="id">
       <wsdl:input message="impl:getCreditCardRequest" name="getCreditCardRequest"/>
       <wsdl:output message="impl:getCreditCardResponse" name="getCreditCardResponse"/>
    </wsdl:operation>
  - <wsdl:operation name="getLoginCount" parameterOrder="id">
       <wsdl:input message="impl:getLoginCountRequest" name="getLoginCountRequest"/>
       <wsdl:output message="impl:getLoginCountResponse" name="getLoginCountResponse"/>
    </wsdl:operation>
  - <wsdl:operation name="getLastName" parameterOrder="id">
       <wsdl:input message="impl:getLastNameRequest" name="getLastNameRequest"/>
       <wsdl:output message="impl:getLastNameResponse" name="getLastNameResponse"/>
    </wsdl:operation>
  - <wsdl:operation name="getFirstName" parameterOrder="id">
       <wsdl:input message="impl:getFirstNameRequest" name="getFirstNameRequest"/>
       <wsdl:output message="impl:getFirstNameResponse" name="getFirstNameResponse"/>
    </wsdl:operation>
```

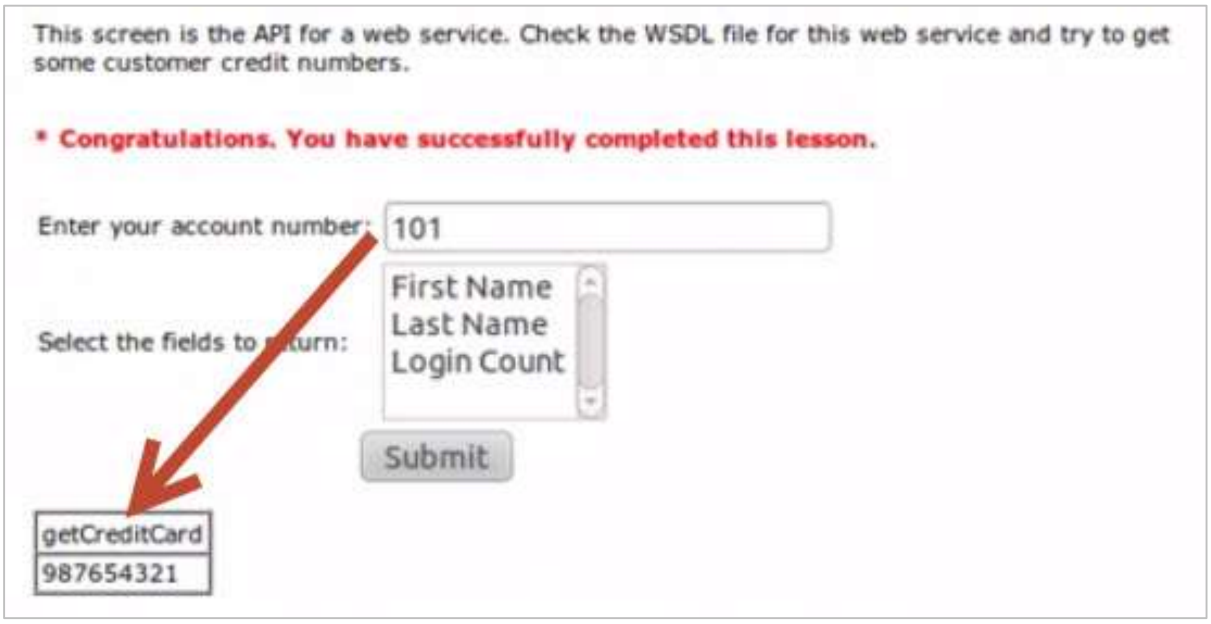
4. Now let us modify the inputs using Burp suite as shown below:


Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=48&menu=1900 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Length: 39
Cookie: JSESSIONID=EC63DA41B54EE7B8000424419D3E9206
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

id=101&field=getCreditCard&SUBMIT=Submit
```

5. We can get the credit card information of other users.



This screen is the API for a web service. Check the WSDL file for this web service and try to get some customer credit numbers.

* Congratulations. You have successfully completed this lesson.

Enter your account number: 101

Select the fields to return:

getCreditCard
987654321

Preventive Mechanisms

- Since SOAP messages are XML-based, all passed credentials have to be converted to text format. Hence one has to be very careful in passing the sensitive information which has to be always encrypted.

- Protecting message integrity by implementing the mechanisms like checksum applied to ensure packet's integrity.

- Protecting message confidentiality - Asymmetric encryption is applied to protect the symmetric session keys, which in many implementations are valid for one communication only and are discarded subsequently.

23. Testing Buffer Overflows

A buffer overflow arises when a program tries to store more data in a temporary data storage area (buffer) than it was intended to hold. Since buffers are created to contain a finite amount of data, the extra information can overflow into adjacent buffers, thus corrupting the valid data held in them.

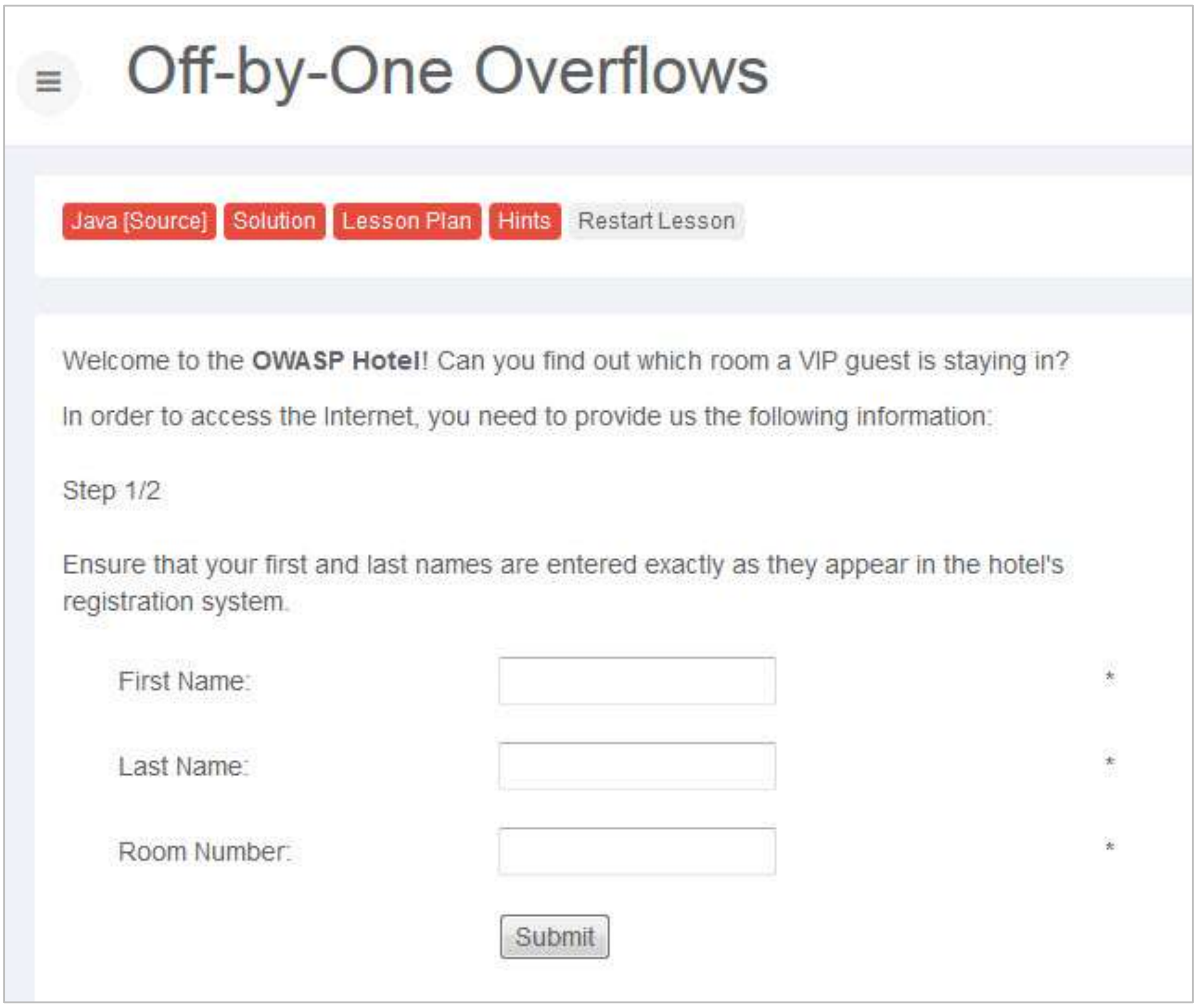
Example

Here is a classic examples of buffer overflow. It demonstrates a simple buffer overflow that is caused by the first scenario in which relies on external data to control its behavior. There is no way to limit the amount of data that user has entered and the behavior of the program depends on the how many characters the user has put inside.

```
...
 char bufr[BUFSIZE];
 gets(bufr);
 ...
```

Hands ON

1. We need to login with name and room number to get the internet access. Here is the scenario snapshot.



Welcome to the **OWASP Hotel!** Can you find out which room a VIP guest is staying in?

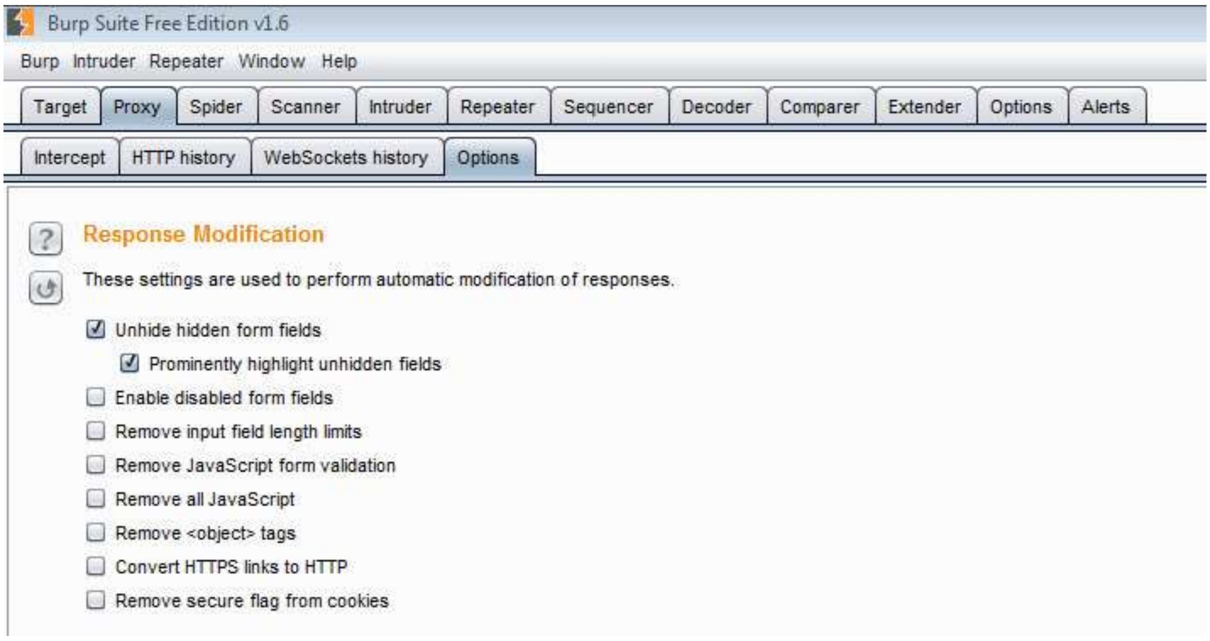
In order to access the Internet, you need to provide us the following information:

Step 1/2

Ensure that your first and last names are entered exactly as they appear in the hotel's registration system.

First Name:		*
Last Name:		*
Room Number:		*

2. We will also enable "Unhide hidden form fields" in Burp Suite as shown below:

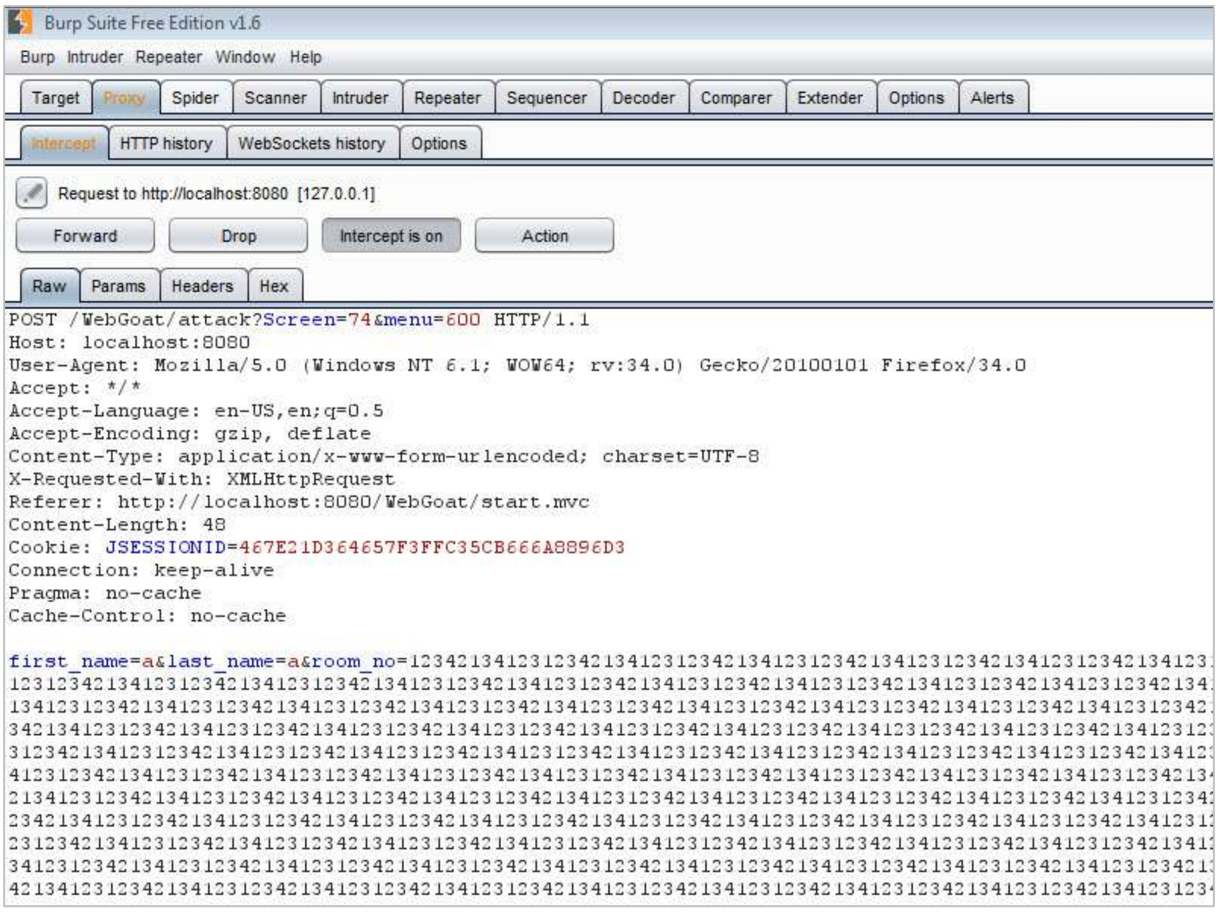

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options

Response Modification

These settings are used to perform automatic modification of responses.

- [x] Unhide hidden form fields
    - [x] Prominently highlight unhidden fields
- [ ] Enable disabled form fields
- [ ] Remove input field length limits
- [ ] Remove JavaScript form validation
- [ ] Remove all JavaScript
- [ ] Remove <object> tags
- [ ] Convert HTTPS links to HTTP
- [ ] Remove secure flag from cookies

3. Now we send an input in name and room number field. We also try and inject a pretty big number in the room number field.

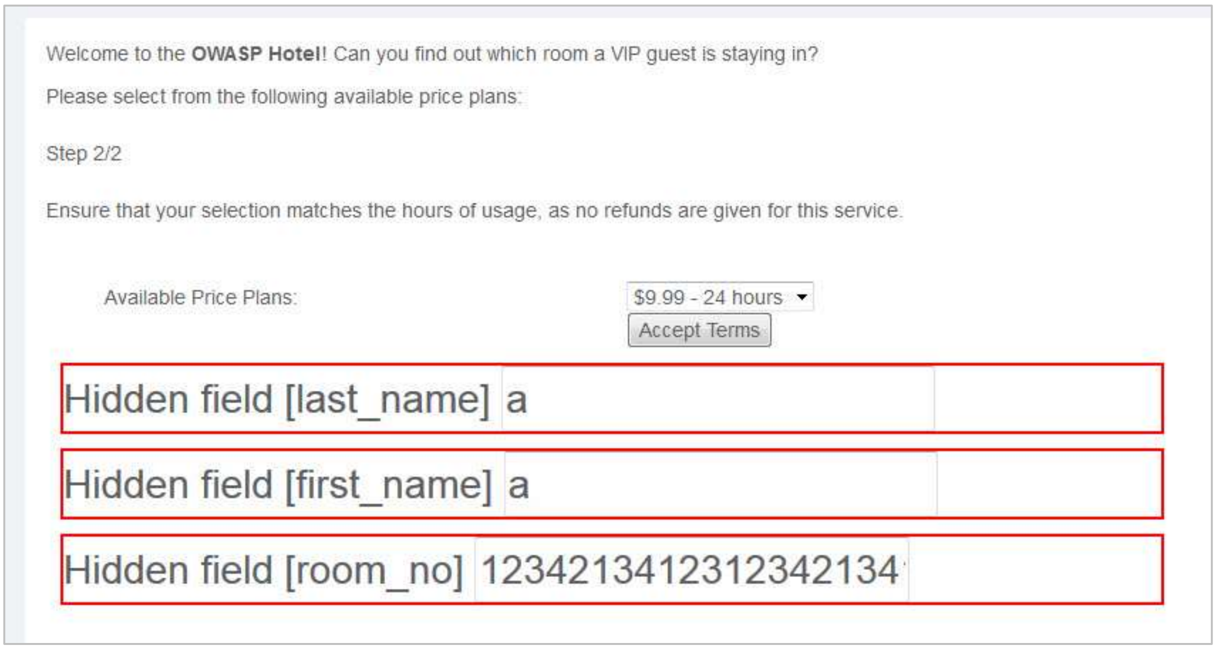

Burp Suite Free Edition v1.6
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Params Headers Hex

Request to http://localhost:8080 [127.0.0.1]

```
POST /WebGoat/attack?Screen=74&menu=600 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Length: 48
Cookie: JSESSIONID=467E21D364657F3FFC35CB666A8896D3
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

first_name=a&last_name=a&room_no=12342134123123421341231234213412312342134123123421341231234213412312342134
12312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231234213412312342
13412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231234213412312
34213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231234213412
31234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231234213
41231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231234
21341231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341231
23421341231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421341
23123421341231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123421
34123123421341231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123123
42134123123421341231234213412312342134123123421341231234213412312342134123123421341231234213412312342134123
```

4. The hidden fields are displayed as shown below. We click accept terms.



Welcome to the **OWASP Hotel**! Can you find out which room a VIP guest is staying in?

Please select from the following available price plans:

Step 2/2

Ensure that your selection matches the hours of usage, as no refunds are given for this service.

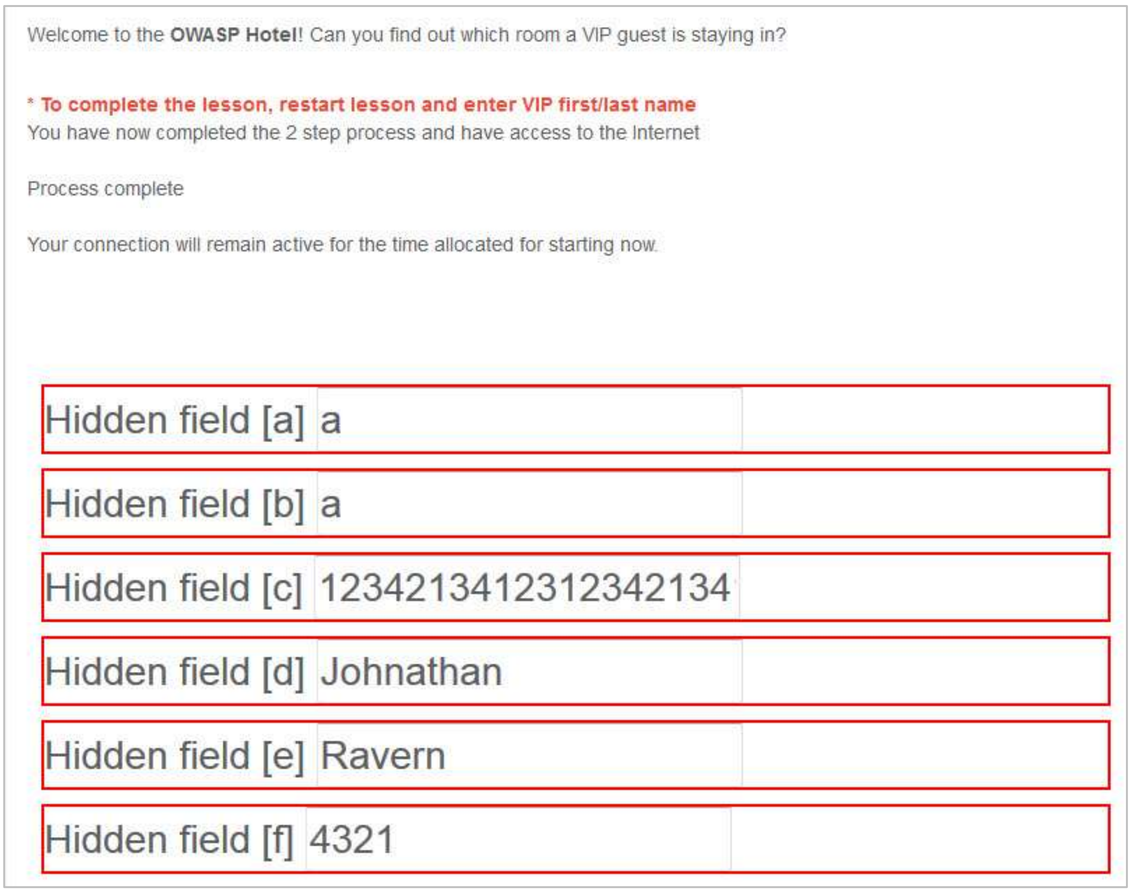
Available Price Plans: $9.99 - 24 hours

Hidden field [last_name] a

Hidden field [first_name] a

Hidden field [room_no] 12342134123123421341

5. The attack is successful such that as a result of buffer overflow, it started reading the adjacent memory locations and displayed to the user as shown below.



Welcome to the **OWASP Hotel**! Can you find out which room a VIP guest is staying in?

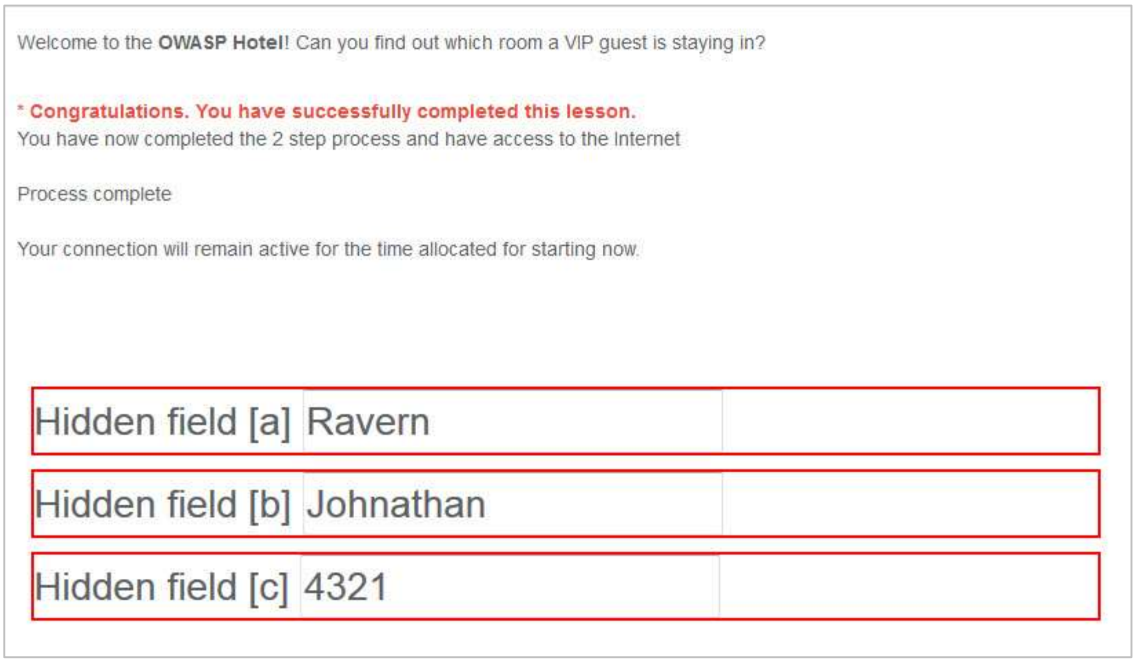
* To complete the lesson, restart lesson and enter VIP first/last name
You have now completed the 2 step process and have access to the Internet

Process complete

Your connection will remain active for the time allocated for starting now.

Hidden field [a]	a
Hidden field [b]	a
Hidden field [c]	123421341231234213‌4
Hidden field [d]	Johnathan
Hidden field [e]	Ravern
Hidden field [f]	4321

6. Now let us login using the data displayed. After logging, the following message is displayed:



Welcome to the **OWASP Hotel**! Can you find out which room a VIP guest is staying in?

* Congratulations. You have successfully completed this lesson.
You have now completed the 2 step process and have access to the Internet

Process complete

Your connection will remain active for the time allocated for starting now.

Hidden field [a]	Ravern
Hidden field [b]	Johnathan
Hidden field [c]	4321

Preventive Mechanisms

- Code Reviewing
- Developer training
- Compiler tools
- Developing Safe functions
- Periodical Scanning

24. Testing Denial of Service

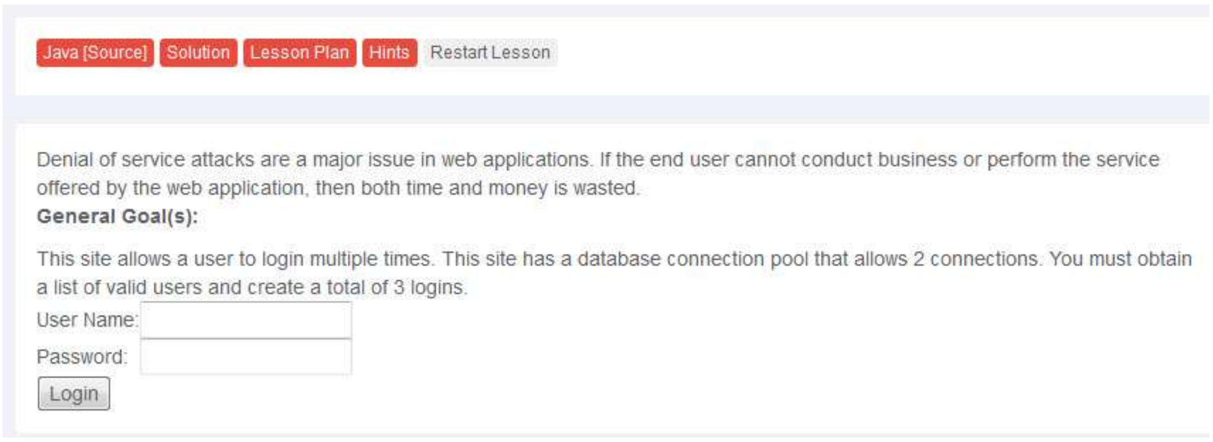
Denial of Service (DoS) attack is an attempt by hackers to make a network resource unavailable. It usually interrupts the host, temporary or indefinitely, which is connected to the internet. These attacks typically target services hosted on mission critical web servers such as banks, credit card payment gateways.

Symptoms of DoS

- Unusually slow network performance.
- Unavailability of a particular web site.
- Inability to access any web site.
- Dramatic increase in the number of spam emails received.
- Long term denial of access to the web or any internet services.
- Unavailability of a particular website.

Hands ON

1. Launch WebGoat and navigate to 'Denial of Service' section. The snapshot of the scenario is given below. We need to login multiple times there by breaching maximum DB thread pool size.



Denial of service attacks are a major issue in web applications. If the end user cannot conduct business or perform the service offered by the web application, then both time and money is wasted.

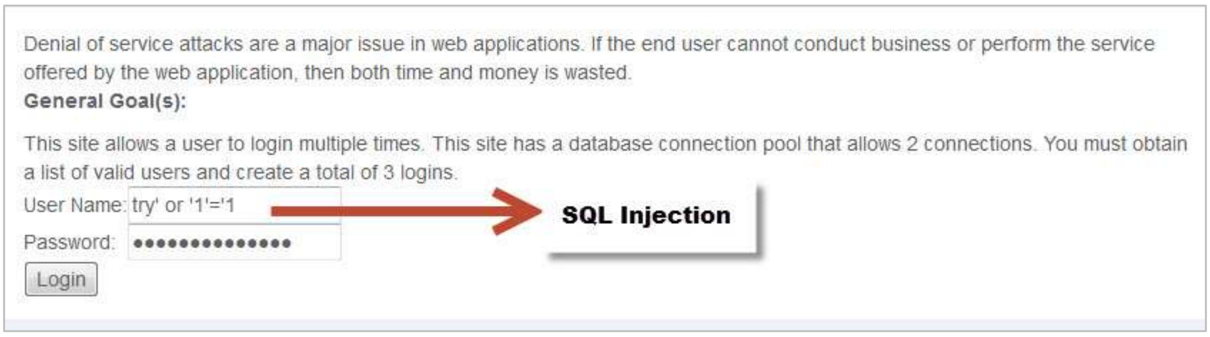
General Goal(s):

This site allows a user to login multiple times. This site has a database connection pool that allows 2 connections. You must obtain a list of valid users and create a total of 3 logins.

User Name:

Password:

2. First we need to get the list of valid logins. We use SQL Injection in this case.



Denial of service attacks are a major issue in web applications. If the end user cannot conduct business or perform the service offered by the web application, then both time and money is wasted.

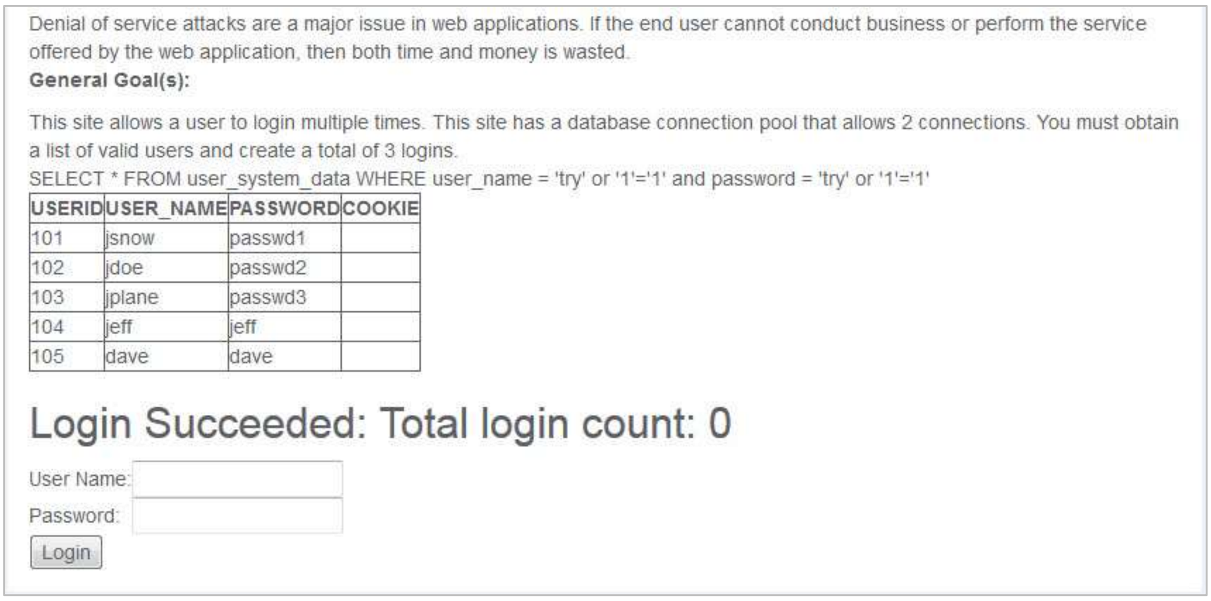
General Goal(s):

This site allows a user to login multiple times. This site has a database connection pool that allows 2 connections. You must obtain a list of valid users and create a total of 3 logins.

User Name: try' or '1'='1

Password: •••••••••••••••

3. If the attempt is successful, then it displays all valid credentials to the user.



Denial of service attacks are a major issue in web applications. If the end user cannot conduct business or perform the service offered by the web application, then both time and money is wasted.

General Goal(s):

This site allows a user to login multiple times. This site has a database connection pool that allows 2 connections. You must obtain a list of valid users and create a total of 3 logins.

SELECT * FROM user_system_data WHERE user_name = 'try' or '1'='1' and password = 'try' or '1'='1'

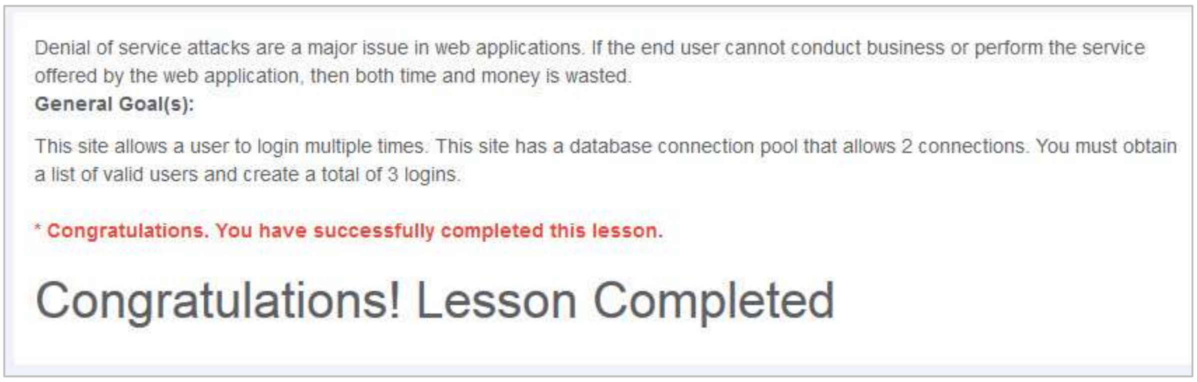
USERID	USER_NAME	PASSWORD	COOKIE
101	jsnow	passwd1	
102	jdoe	passwd2	
103	jplane	passwd3	
104	jeff	jeff	
105	dave	dave	

Login Succeeded: Total login count: 0

User Name:

Password:

4. Now login with each one of these user in at least 3 different sessions in order to make the DoS attack successful. As we know that DB connection can handle only two threads, by using all logins it will create three threads which makes the attack successful.



Denial of service attacks are a major issue in web applications. If the end user cannot conduct business or perform the service offered by the web application, then both time and money is wasted.

General Goal(s):

This site allows a user to login multiple times. This site has a database connection pool that allows 2 connections. You must obtain a list of valid users and create a total of 3 logins.

* Congratulations. You have successfully completed this lesson.

Congratulations! Lesson Completed

Preventive Mechanisms

- Perform thorough input validations. Avoid highly CPU consuming operations.
- It is better to separate data disks from system disks.

25. Malicious File Execution

Developers often directly use or concatenate potentially vulnerable input with file or assume that input files are genuine. When the data is not checked properly, this can lead to the vulnerable content being processed or invoked by the web server.

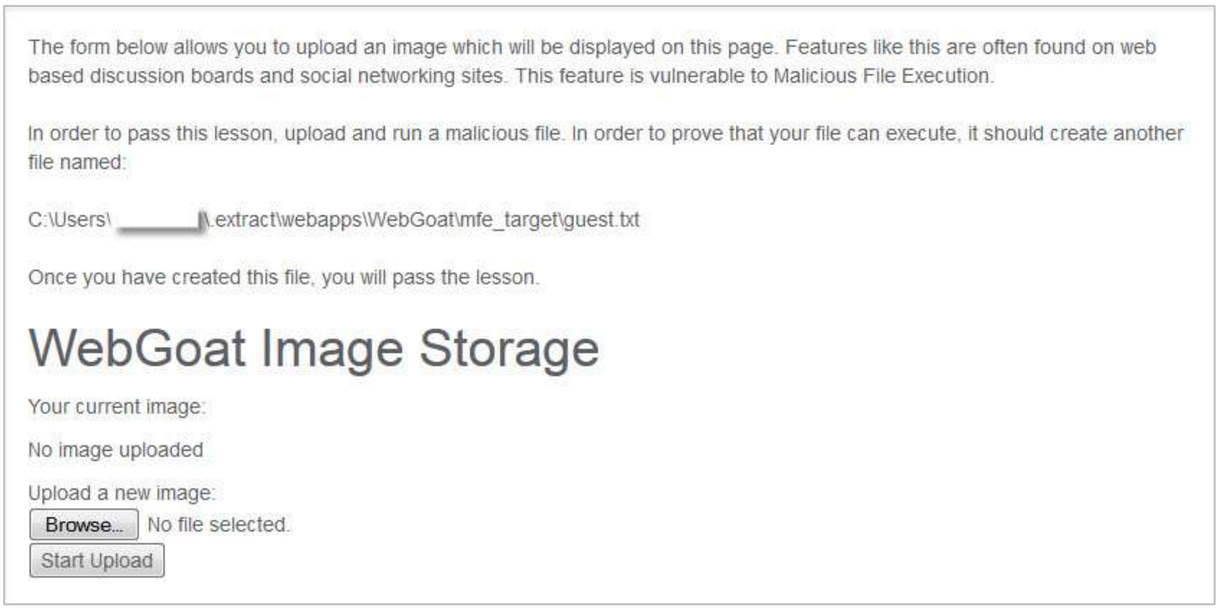
Example

Some of the classic examples include:

- Upload .jsp file into web tree.
- Upload .gif to be resized.
- Upload huge files.
- Upload file containing tags.
- Upload .exe file into web tree.

Hands ON

1. Launch WebGoat and navigate to Malicious file execution section. The snapshot of the scenario is given below:



The form below allows you to upload an image which will be displayed on this page. Features like this are often found on web based discussion boards and social networking sites. This feature is vulnerable to Malicious File Execution.

In order to pass this lesson, upload and run a malicious file. In order to prove that your file can execute, it should create another file named:

C:\Users\[redacted]\.extract\webapps\WebGoat\mfe_target\guest.txt

Once you have created this file, you will pass the lesson.

WebGoat Image Storage

Your current image:

No image uploaded

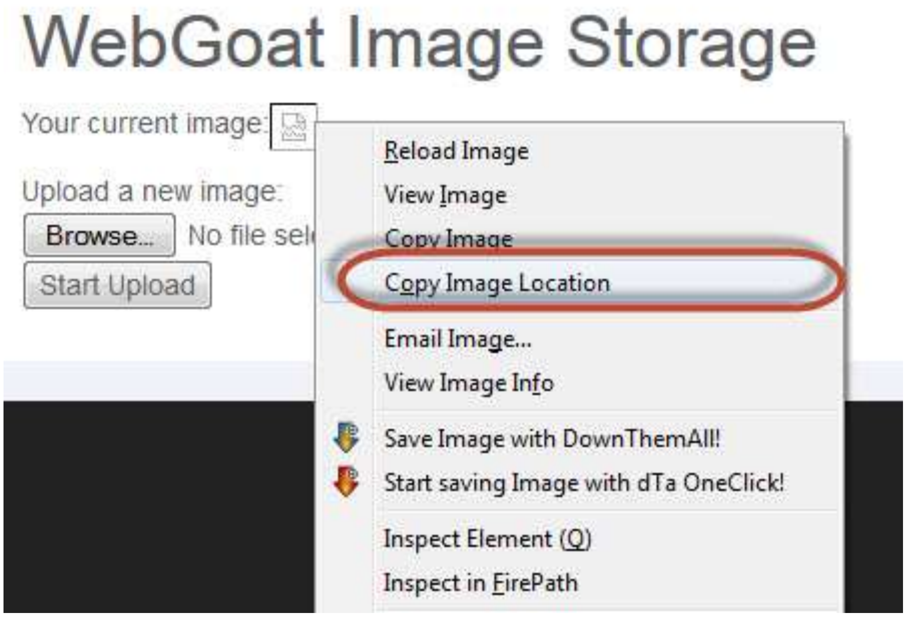
Upload a new image:

2. In order to complete this lesson, we need to upload guest.txt in the above said location.

3. Let us create a jsp file such that the guest.txt file is created on executing the jsp. The Naming of the jsp has no role to play in this context as we are executing the content of the jsp file.

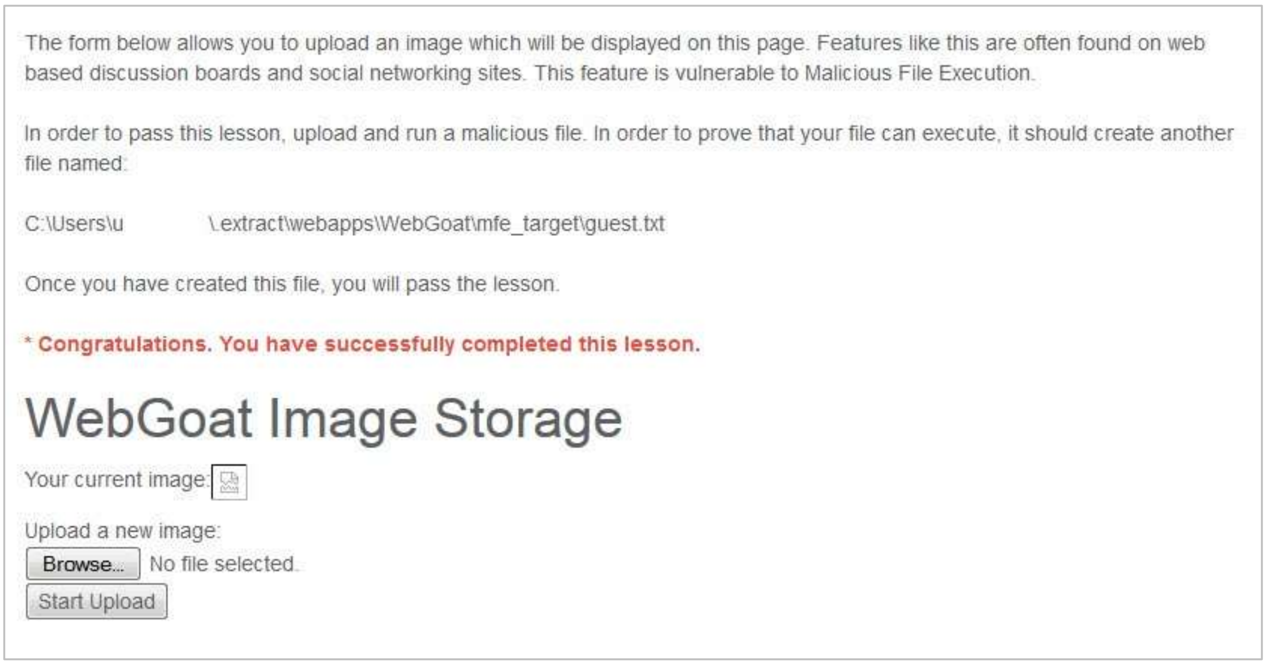
```
<HTML> <% java.io.File file = new
java.io.File("C:\\Users\\username$\\.extract\\webapps\\WebGoat\\mfe_target\\gue
st.txt"); file.createNewFile(); %> </HTML>
```

4. Now upload the jsp file and copy the link location of the same after upload. The upload is expecting an image, but we are uploading a jsp.



5. By navigating to the jsp file, there will not be any message to the user.

6. Now refresh the session where you have uploaded the jsp file and you will get the message saying, "* Congratulations. You have successfully completed the lesson".



The form below allows you to upload an image which will be displayed on this page. Features like this are often found on web based discussion boards and social networking sites. This feature is vulnerable to Malicious File Execution.

In order to pass this lesson, upload and run a malicious file. In order to prove that your file can execute, it should create another file named:

C:\Users\u \.extract\webapps\WebGoat\mfe_target\guest.txt

Once you have created this file, you will pass the lesson.

* Congratulations. You have successfully completed this lesson.

Preventive Mechanisms

- Secure websites using website permissions.
- Adopt countermeasures for web application security.
- Understand the Built-In user and group accounts in IIS 7.0.

26. Security Testing – Automation Tools

There are various tools available to perform security testing of an application. There are few tools that can perform end-to-end security testing while some are dedicated to spot a particular type of flaw in the system.

Open Source Tools

Some open source security testing tools are as given:

S. No.	Tool Name
1	**Zed Attack Proxy** Provides Automated Scanners and other tools for spotting security flaws. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
2	**OWASP WebScarab** Developed in Java for Analysing Http and Https requests. https://www.owasp.org/index.php/OWASP_WebScarab_Project
3	**OWASP Mantra** Supports multi-lingual security testing framework https://www.owasp.org/index.php/OWASP_Mantra_-_Security_Framework
4	**Burp Proxy** Tool for Intercepting & Modyfying traffic and works with work with custom SSL certificates. http://www.portswigger.net/Burp/
5	**Firefox Tamper Data** Use tamperdata to view and modify HTTP/HTTPS headers and post parameters https://addons.mozilla.org/en-US/firefox/addon/tamper-data/
6	**Firefox Web Developer Tools** The Web Developer extension adds various web developer tools to the browser. https://addons.mozilla.org/en-US/firefox/addon/web-developer/
7	**Cookie Editor** Lets user to add, delete, edit, search, protect and block cookies https://chrome.google.com/webstore/detail/fngmhnnpilhplaeedifhccceomclgfbg?hl=en-US

Specific Tool Sets

The following tools can help us spot a particular type of vulnerability in the system:

S. No.	Link
1	**DOMinator Pro - Testing for DOM XSS** https://dominator.mindedsecurity.com/
2	**OWASP SQLiX - SQL Injection** https://www.owasp.org/index.php/Category:OWASP_SQLiX_Project
3	**Sqlninja - SQL Injection** http://sqlninja.sourceforge.net/
4	**SQLInjector - SQL Injection** http://sourceforge.net/projects/safe3si/
5	**sqlpowerinjector - SQL Injection** http://www.sqlpowerinjector.com/
6	**SSL Digger - Testing SSL** http://www.mcafee.com/us/downloads/free-tools/ssldigger.aspx
7	**THC-Hydra - Brute Force Password** https://www.thc.org/thc-hydra/
8	**Brutus - Brute Force Password** http://www.hoobie.net/brutus/
9	**Ncat - Brute Force Password** http://nmap.org/ncat/
10	**OllyDbg - Testing Buffer Overflow** http://www.ollydbg.de/
11	**Spike - Testing Buffer Overflow** http://www.immunitysec.com/downloads/SPIKE2.9.tgz
12	**Metasploit - Testing Buffer Overflow** http://www.metasploit.com/

Commercial Black Box Testing tools

Here are some of the commercial black box testing tools that help us spot security issues in the applications that we develop.

S. No.	Tool
1	**NGSSQuirreL** https://www.nccgroup.com/en/our-services/security-consulting/information-security-software/squirrel-vulnerability-scanner/
2	**IBM AppScan** http://www-01.ibm.com/software/awdtools/appscan/
3	**Acunetix Web Vulnerability Scanner** http://www.acunetix.com/
4	**NTOSpider** http://www.ntobjectives.com/products/ntospider.php
5	**SOAP UI** http://www.soapui.org/Security/getting-started.html
6	**Netsparker** http://www.mavitunasecurity.com/netsparker/
7	**HP WebInspect** http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect

Free Source Code Analyzers

S. No.	Tool
1	**OWASP Orizon** https://www.owasp.org/index.php/Category:OWASP_Orizon_Project
2	**OWASP O2** https://www.owasp.org/index.php/OWASP_O2_Platform
3	**SearchDiggity**

	http://www.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/
4	**FXCOP** https://www.owasp.org/index.php/FxCop
5	**Splint** http://splint.org/
6	**Boon** http://www.cs.berkeley.edu/~daw/boon/
7	**W3af** http://w3af.org/
8	**FlawFinder** http://www.dwheeler.com/flawfinder/
9	**FindBugs** http://findbugs.sourceforge.net/

Commercial Source Code Analyzers

These analyzers examine, detect, and report the weaknesses in the source code, which are prone to vulnerabilities:

S. No.	Tool
1	**Parasoft C/C++ test** http://www.parasoft.com/cpptest/testing_malacious_file_execution.htm
2	**HP Fortify** http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer
3	Appscan http://www-01.ibm.com/software/rational/products/appscan/source/
4	**Veracode** http://www.veracode.com
5	**Armorize CodeSecure** http://www.armorize.com/codesecure/
6	**GrammaTech** http://www.grammatech.com/