Security Testing

protect the data and maintain functionality

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

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

About the Tutorial i

Audience i

Prerequisites i

Copyright & Disclaimer i

Table of Contents ii

1. SECURITY TESTING – OVERVIEW 1

What is Security Testing? 1

Example 1

2. SECURITY TESTING – PROCESS 2

Penetration Test – Workflow 2

Footprinting 3

Footprinting – Steps 3

Scanning 4

Enumeration 5

Exploitation 6

3. SECURITY TESTING – MALICIOUS SOFTWARE 8

Malwares 8

Preventive Measures 8

Anti-Malware Software 9

4. SECURITY TESTING – HTTP PROTOCOL BASICS 10

HTTP Protocol 10

Basic Features 10

Architecture 11

HTTP Parameters 11

HTTP Messages **14**

HTTP Requests **16**

HTTP Responses **20**

HTTP Methods **23**

HTTP Status Codes **30**

HTTP Header Fields **33**

Client Request Headers **37**

Server Response Headers **44**

Entity Headers **47**

HTTP Security **50**

5. SECURITY TESTING – HTTPS PROTOCOL BASICS 53

When is HTTPS Required? **53**

Basic Working of HTTPS **53**

6. SECURITY TESTING – ENCODING AND DECODING 55

What is Encoding and Decoding? **55**

7. SECURITY TESTING – CRYPTOGRAPHY 61

What is Cryptography? **61**

How Encryption Works? **61**

Cryptography Techniques **61**

8. SECURITY TESTING – SAME ORIGIN POLICY 63

What is Same Origin Policy? **63**

Example **63**

Same Origin policy Exceptions for IE **63**

9. SECURITY TESTING – TESTING COOKIES 64

What is a Cookie? 64

Properties of Cookies 64

Cookie Contents 64

Types of Cookies 64

Testing Cookies 65

Viewing and Editing Cookies 65

10. SECURITY TESTING – HACKING WEB APPLICATIONS 67

Web Application - PenTesting Methodologies 67

OWASP Top 10 67

Application - Hands On 68

Web Proxy 69

Configuring Burp Suite 70

11. SECURITY TESTING – TESTING INJECTION 73

Web Application - Injection 73

Examples 74

Preventing SQL Injection 76

12. SECURITY TESTING – TESTING BROKEN AUTHENTICATION 77

Preventing Mechanisms 80

13. SECURITY TESTING – TESING CROSS-SITE SCRIPTING 81

Types of XSS 81

Example 82

Preventive Mechanisms 85

14. SECURITY TESTING – INSECURE DIRECT OBJECT REFERENCES 86

Example **86**

Preventive Mechanisms **89**

15. SECURITY TESTING – SECURITY MISCONFIGURATION 90

Example **90**

Preventive Mechanisms **92**

16. SECURITY TESTING – TESTING SENSITIVE DATA EXPOSURE 93

Example **93**

Preventive Mechanisms **94**

17. SECURITY TESTING – MISSING FUNCTION LEVEL ACCESS CONTROL 95

Example **95**

Preventive Mechanisms **97**

18. SECURITY TESTING – CROSS-SITE REQUEST FORGERY (CSRF) 98

Example **98**

Preventive Mechanisms **100**

19. SECURITY TESTING – COMPONENTS WITH VULNERABILITIES 101

Example **101**

Preventive Mechanisms **102**

20. SECURITY TESTING – UNVALIDATED REDIRECTS AND FORWARDS 103

Example **103**

Preventive Mechanisms **104**

21. SECURITY TESTING – AJAX SECURITY 105

Example **105**

Preventive Mechanisms **109**

22. SECURITY TESTING – WEB SERVICE SECURITY 110

Preventive Mechanisms **112**

23. SECURITY TESTING – TESTING BUFFER OVERFLOWS 113

Example **113**

Preventive Mechanisms **117**

24. SECURITY TESTING – TESTING DENIAL OF SERVICE 118

Symptoms of DoS **118**

Preventive Mechanisms **119**

25. SECURITY TESTING – MALICIOUS FILE EXECUTION 120

Example **120**

Preventive Mechanisms **122**

26. SECURITY TESTING – AUTOMATION TOOLS 123

Open Source Tools **123**

Specific Tool Sets **124**

Commercial Black Box Testing tools **125**

Free Source Code Analyzers **125**

Commercial Source Code Analyzers **127**

1. Security Testing – 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.
3. Log out of the web application.
4.
5. Click the BACK button of the browser.
6.
7. 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**.

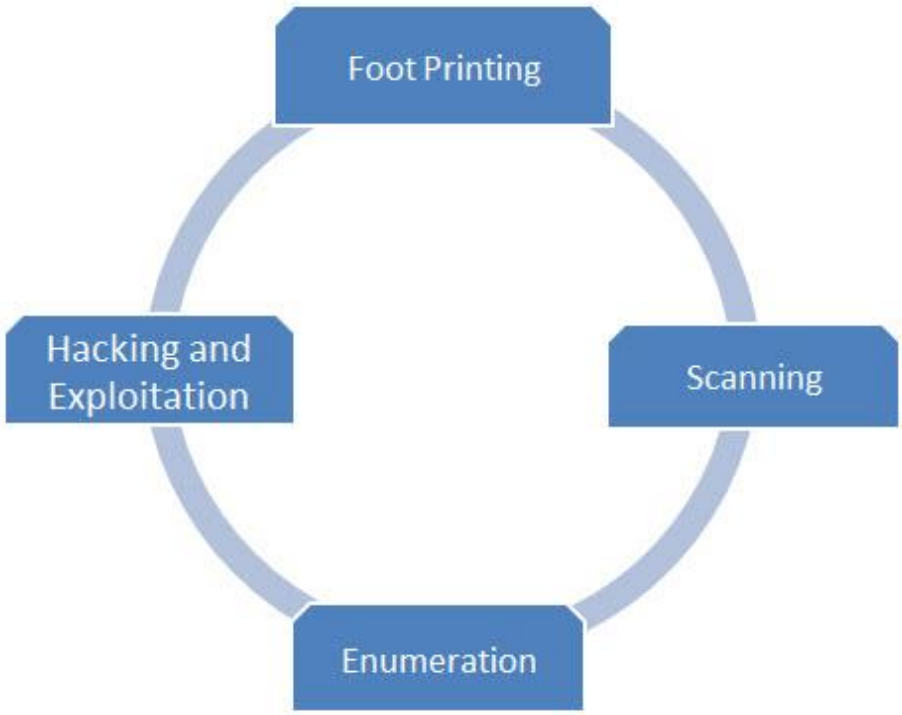
Penetration test is done in phases and here in this chapter, we will discuss the complete process. Proper documentation should be done in each phase so that all the steps necessary to reproduce the attack are available readily. The documentation also serves as the basis for the detailed report customers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

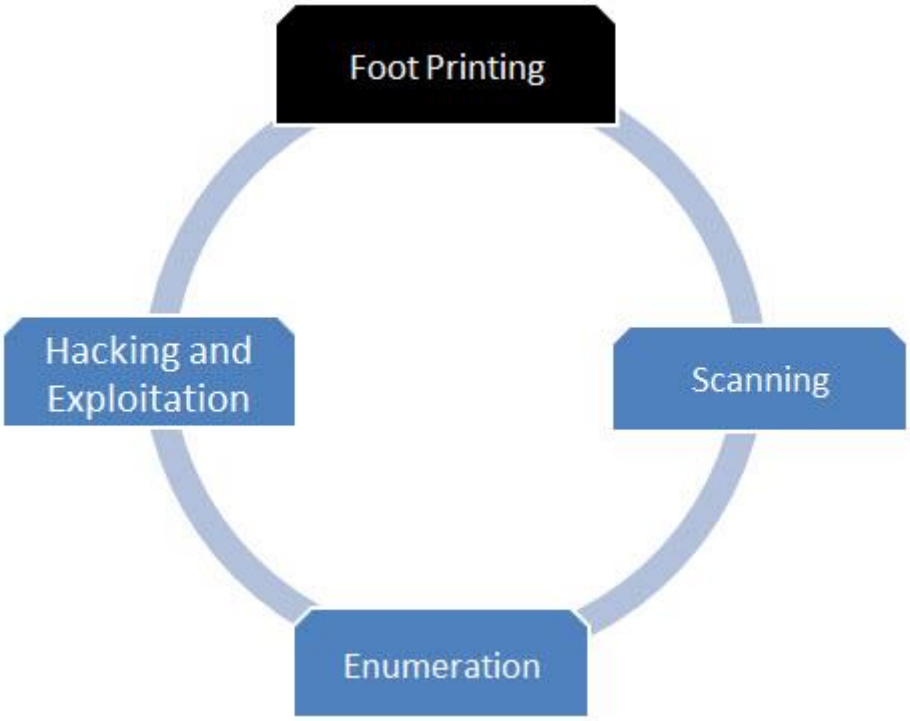
- Whois
- SmartWhois
- NsLookup
- Sam Spade

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

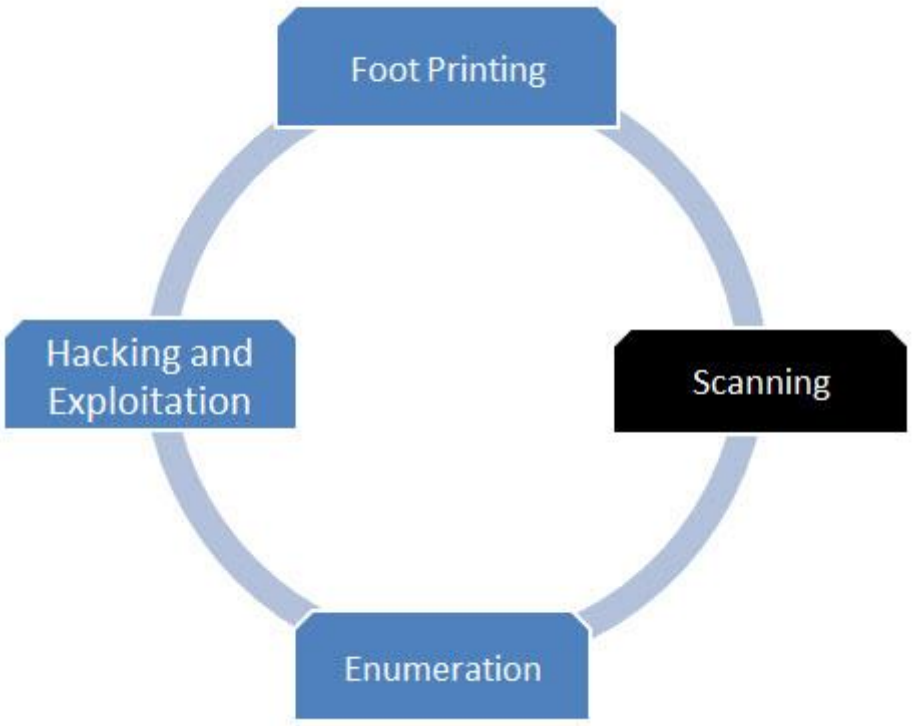
Scanning is the second step that is performed after footprinting. It involves scanning open ports, fingerprinting the operating system, and uncovering services on ports. The ultimate goal of scanning is to find open ports through external or internal network scanning, pinging machines, determining network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

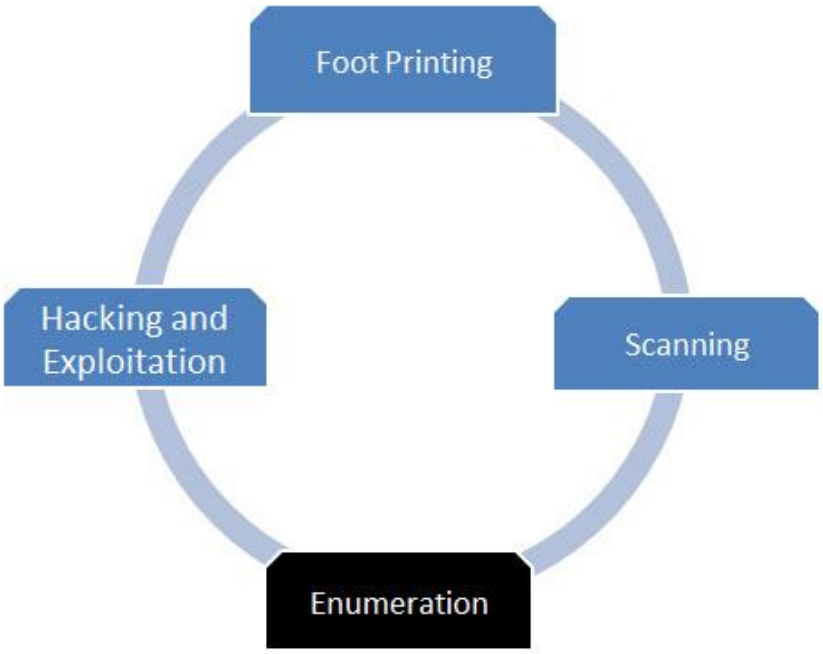
Enumeration is the next step after scanning. The goal of enumeration is to get a complete picture of the target. In this phase, a penetration tester tries to identify valid user accounts or poorly-protected shared resources using active connections to systems.

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

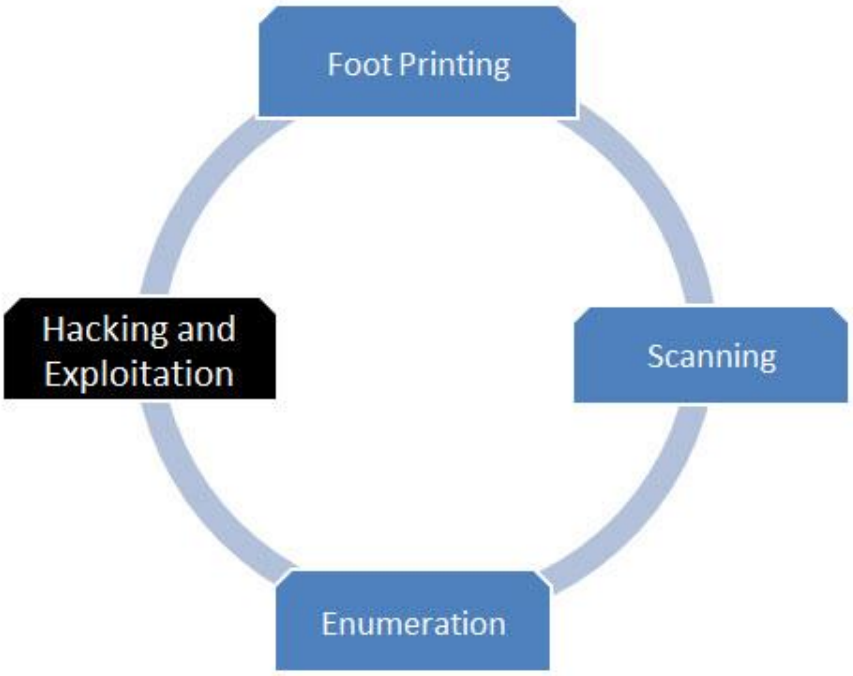
The types of exploitation are segregated into three different categories:

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. Security Testing – 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. Security Testing – 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.

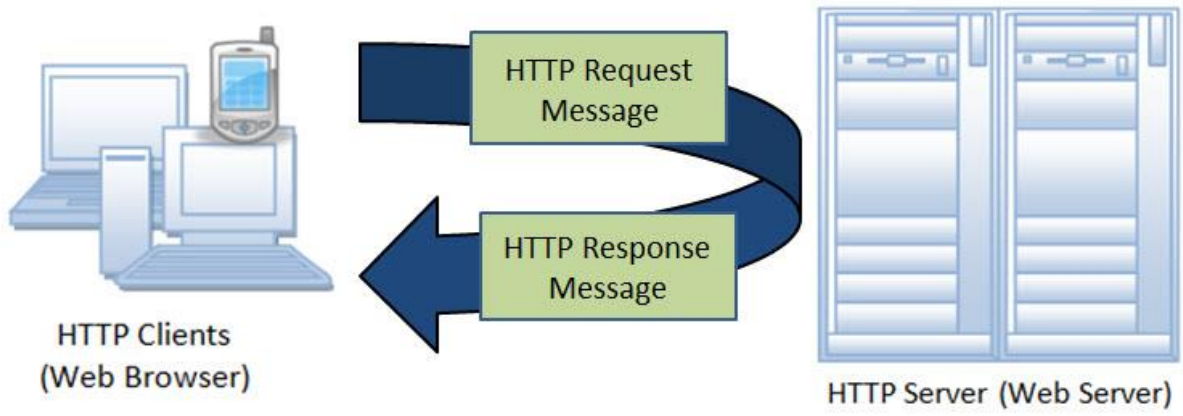
- **HTTP is media independent:** Any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. This is required for client as well as server to specify the content type using appropriate MIME-type.

- **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 nobody 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 the 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>
```