HTTP BASICS

What is HTTP?

HTTP is a stateless protocol. Protocols are a set of rules which has to be followed to maintain a strict boundary of instructions for a networking system. HTTP stands for Hypertext Transfer Protocol which in itself suggests it is a protocol based out of text commands and used to transfer data via these pre-set of command instructions. The commands itself are test supported and therefore it makes easy for a networking point of view to collaborate, distribute, interact and communicate among hypertext networking systems. Statelessness of HTTP protocol means the protocol supports being on different state from the original one. This means if a request is sent from a client to a server via a medium; this newly introduced medium could tamper with the request methods and then send the newly formed request to the server. Now this has security concerns; which is why this paper is all about. It's to know the HTTP protocol closely.

The World Wide Web largely depends on the HTTP protocol, as the protocol itself is light, fast, efficient and dependent on use of status codes, headers and request methods. This use of dependent headers, status code and request methods makes the HTTP protocol suited the best for any packet based communication in-between a network, group of network or an isolated network. The study of HTTP is a must for any web application security enthusiast as later for HTTP tampering, HTTP based attacks, HTTP CR/LF attacks, and other HTTP attacks; HTTP based required knowledge would be needed. It's to be added that HTTP is an application level protocol and works at the application level of the OSI model. Find the references of HTTP below.

HTTP is said to be asymmetric request-response client-server protocol. It's also a 'pull' based protocol which means the client pulls out information from the server rather than the server 'pushing' the information back to the client. To study HTTP protocols, we must have a basic idea of an URL or Uniform Resource Locator. There are URI and URN too. That comes later.

Terminology:

Connection: A transport layer virtual circuit established between two programs for the purpose of communication.

Message: The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax (of the referenced RFC) and transmitted via the connection.

Request: An HTTP request message.

Response: An HTTP response message.

Resource: A network data object or service that can be identified by a URI. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

Entity: The information transferred as the payload of a request or response. An entity consists of meta information in the form of entity-header fields and content in the form of an entity-body.

Representation: An entity included with a response that is subject to content negotiation. There may exist multiple representations associated with a particular response status.
Content negotiation: The mechanism for selecting the appropriate representation when servicing a request. The representation of entities in any response can be negotiated (including error responses).

Variant: A resource may have one, or more than one, representation(s) associated with it at any given instant. Each of these representations is termed a `variant'. Use of the term `variant' does not necessarily imply that the resource is subject to content negotiation.

Client: A program that establishes connections for the purpose of sending requests.
User agent: The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.

Server: An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

Origin server: The server on which a given resource resides or is to be created.

Proxy: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy MUST implement both the client and server requirements of this specification. A "transparent proxy" is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification. A "non-transparent proxy" is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering. Except where either transparent or non-transparent behavior is explicitly stated, the HTTP proxy requirements apply to both types of proxies.

Gateway: A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the

requested resource; the requesting client may not be aware that it is communicating with a gateway.

Tunnel: An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

Cache: A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel.

Cacheable: A response is cacheable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cacheability of HTTP. Even if a resource is cacheable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

First-hand: A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps via one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

Explicit expiration time: The time at which the origin server intends that an entity should no longer be returned by a cache without further validation.

Heuristic expiration time: An expiration time assigned by a cache when no explicit expiration time is available.

Age: The age of a response is the time since it was sent by, or successfully validated with, the origin server.

Freshness lifetime: The length of time between the generation of a response and its expiration time.

Fresh: A response is fresh if its age has not yet exceeded its freshness lifetime.

Stale: A response is stale if its age has passed its freshness lifetime.

Semantically transparent: A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response (except for hop-by-hop headers) that it would have received had its request been handled directly by the origin server.

Validator: A protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.

Upstream/downstream: Upstream and downstream describe the flow of a message: all messages flow from upstream to downstream.

Inbound/outbound: Inbound and outbound refer to the request and response paths for messages: "inbound" means "traveling toward the origin server", and "outbound" means "traveling toward the user agent"

What is an URL?

URL is short for Uniform Resource Locator. A generic URL has four parts to it:

1. **Protocol**
2. **Hostname**
3. **Port**
4. **Path and file**

To breakdown all of the mentioned 4 entities of an URL, let's take how an actual URL would look like.

http://www.mytestsite.com:80/destination_path/index.htm

Here:

1. Http is the protocol.

2. www.mytestsite.com is the host.

3. 80 is the port, by default if it's not mentioned, 80 is taken as default.
4. Destination_path/index.htm is the destination directory with a file within.

Broadly, HTTP is the stateless protocol used to communicate or request a resource from the server from the client. HTTP works on TCP suite and 80 is the default port number for the server to listen on which acts as a HTTP server. Hostname is www.mytestsite.com which is again the DNS host and it could be an IP too such as for localhost, it could be something like "192.168.78.11". The rest is 'path and the filename' of the requested resource on the server. The server may have directories and sub-directories and end up with a particular file that is being requested. In this case it's index.htm. There could had been an index.htm file under the document root directory of the server too, however here for illustration purposes, the 'directory' path is shown for conveying directories could be included as well. HTTP is an application level protocol.

Application level protocols.

There are different application level protocols for the client and the server to talk to each other. These protocols are varied and few of them are mentioned below for reference purposes only.

1. SMTP: Smart Mail Transfer Protocol.
2. GMTP: Group Mail Transfer Protocol.
3. FTP: File Transfer Protocol.
4. TFTP: Trivial File Transfer Protocol.
5. ARP: Address Resolution Protocol.
6. RARP: Reverse Address Resolution protocol.
7. Telnet: Remote Terminal Access Protocol.
8. ADC
9. AFP
10. BACnet
11. BitTorrent
12. BGP: Border Gateway Protocol.
13. BOOTP: Bootstrap Protocol.
14. CAMEL
15. DiCOM: An Authentication, Authorization and Accounting Protocol.

16. DICT: Dictionary Protocol.
17. DNS: domain Name System.
18. DSM-CC: Digital Storage Media Command and Control.
19. DNSP: Dynamic Host Control Protocol.
20. ED2K: A peer to peer protocol.
21. Finger: User profile Information Protocol.
22. Gnutella: Peer to peer file swapping protocol.
23. Gopher: Hierarchical Hyperlinkable protocol.
24. HTTP: Hypertext Transfer Protocol.
25. HTTPS: Hypertext Transfer Protocol Secure.
26. IMAP: Internet Message Access Protocol.
27. IRC: Internet Relay Chat.