

## Capitolo 0.5

### Internet

Nel World Wide Web ogni risorsa viene identificata univocamente da un URL (Uniform Resource Locator).

Per risorsa intendiamo un qualsiasi insieme di dati o informazioni che abbiano senso. Immagini, paragrafi testuali, video, audio, pagine web, risultati dell'elaborazione di un programma sono tutti esempi di risorsa. Wikipedia definisce le "risorse sul Web" come "tutte le fonti di informazioni e servizi disponibili in Rete, identificate dall'URL e fisicamente presenti e accessibili su web server tramite web browser dell'host client."

Se questa definizione non è chiara, sarà utile fare un ripasso del modello client-server

Illustrazione formale:

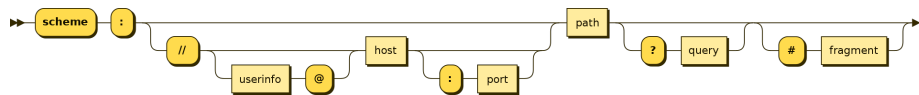


Figure 1: URL formale

Esempio pratico:

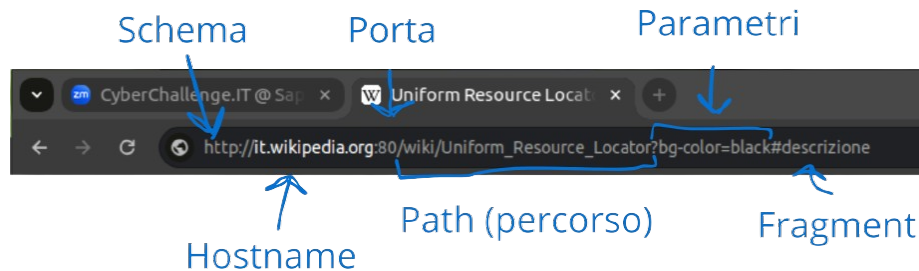


Figure 2: URL pratico

*Nota: la porta di default - quindi il valore valido se non specificato - è l'80, il fragment è di default l'inizio pagina. Le query string, o parametri, sono generalmente facoltativi. Le userinfo, ovvero username e password, non mostrati nell'esempio, sono tipici di protocolli diversi da quelli che vediamo usualmente nel browser, come FTP*

### URL-encoding

Per evitare che nell'URL siano presenti caratteri riservati che potrebbero portare ad un'interpretazione indesiderata da parte del browser, viene usato l'URL

encoding, ufficialmente percent-encoding. Se usati per un attacco è quindi utile ricordarsi di codificare gli URL in modo che nessuna parte del “vettore d’attacco”/payload che abbiamo preparato vada persa.

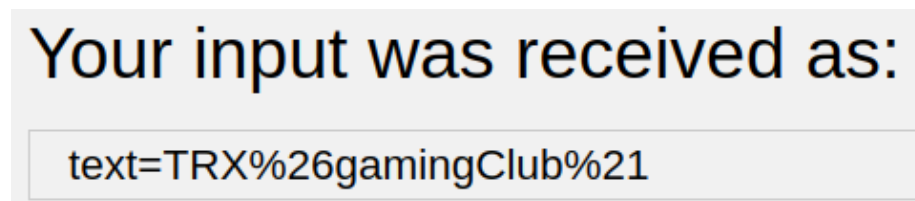
Prima dell’URL-encoding:



A screenshot of a web form. It consists of a rectangular text input field on the left and a rectangular button labeled "Submit" on the right. The text input field contains the string "TRX&gamingClub!".

Figure 3: URL pre-encoding

Dopo l’URL-encoding (cosa riceve il server):



A screenshot of a confirmation message. It features a large heading "Your input was received as:" in a bold, sans-serif font. Below the heading is a light gray rectangular box containing the text "text=TRX%26gamingClub%21".

Figure 4: URL post-encoding

Da notare come solo il testo che può essere direttamente controllato dall’utente venga URL-encodato.

Ciò che viene fatto è una conversione dal carattere riservato alla sua rappresentazione ASCII esadecimale preceduta da un %.

## HTTP

l’HyperText Transfer Protocol, HTTP, è un protocollo stateless, ovvero ogni richiesta è indipendente dalle richieste precedenti. Le due fasi previste sono l’HTTP request (il client fa una richiesta al server) e l’HTTP response (il server risponde).

In generale, ogni volta che il client ha bisogno di richiedere una risorsa al server, comunica utilizzando HTTP. Questo significa che per ogni risorsa che vorrai visualizzare, il tuo dispositivo dovrà effettuare un’HTTP request, e ricevere dal server un’HTTP response.

Esempio di HTTP request:

Esempio di HTTP response:

In caso fosse necessario uno scambio di informazioni al di fuori del contesto degli header e delle query, queste possono essere incluse nel corpo del messaggio (body message). Nell’esempio di richiesta mostrato, apparirebbero “sotto” agli header. La struttura del body message varia a seconda della tecnologia utilizzata

Method Path Version

```

1 GET /ctf_hscf2023/2023/06/12/flagshop.html HTTP/2
2 Host: theromanxpl0.it
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:122.0) Gecko/20100101
  Firefox/122.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: https://theromanxpl0.it/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-User: ?1
13 Te: trailers
14

```

Riga della richiesta

Header(s)

Figure 5: HTTP request

Version Status code

```

1 HTTP/2 200 OK
2 Server: GitHub.com
3 Content-Type: text/html; charset=utf-8
4 Last-Modified: Tue, 12 Dec 2023 21:06:45 GMT
5 Access-Control-Allow-Origin: *
6 Etag: W/"6578cb65-9f1b"
7 Expires: Mon, 19 Feb 2024 13:37:36 GMT
8 Cache-Control: max-age=600
9 X-Proxy-Cache: MISS
10 X-Github-Request-Id: DCE2:0E92:4902CC2:4A55098:65D35748
11 Accept-Ranges: bytes
12 Date: Mon, 19 Feb 2024 13:27:36 GMT
13 Via: 1.1 varnish
14 Age: 0
15 X-Served-By: cache-fco2270027-FC0
16 X-Cache: MISS
17 X-Cache-Hits: 0
18 X-Timer: S1708349257.866990,VS0,VE131
19 Vary: Accept-Encoding
20 X-Fastly-Request-Id: 50d95ffd4ba3bc9b6aec2dc49cdce14babf193ec
21 Content-Length: 40731
22
23 <!DOCTYPE html>
24 <html lang="en-us">

```

Riga dello stato

Header(s)

Body message

Figure 6: HTTP response

dallo specifico sito, ed è comunque facilmente individuabile durante l'esperienza pratica.

#### Metodi HTTP comuni:

- **GET** (ottiene la risorsa o informazioni su di essa)
- **POST** (azione/invia dati alla risorsa)
- **HEAD** (GET senza body message)
- **TRACE** (diagnostica)
- **OPTIONS** (visualizza metodi disponibili)
- **PUT** (crea nuova risorsa)
- **DELETE** (elimina risorsa specificata)
- **CONNECT** (crea un tunnel in caso di proxy)
- **PATCH** (modifica la risorsa)

#### Header HTTP comuni:

##### Request

- **Accept:** Definisce i MIME type che il client accetterà dal server, in ordine di preferenza. Ad esempio, **Accept: application/json, text/html** indica che il client preferisce ricevere risposte in JSON, ma le accetta anche in HTML.
- **User-Agent:** Identifica il browser e/o il client che sta effettuando la richiesta.
- **Authorization:** Usato per l'invio di credenziali, utile quando si prova ad accedere ad una risorsa protetta.
- **Cookie:** Usato per inviare al server cookie precedentemente memorizzati. Utile per personalizzare l'esperienza dell'utente e "combattere" i limiti della natura stateless del protocollo HTTP.
- **Content-Type:** Definisce il MIME type del contenuto del request body.

##### Response

- **Content-Type:** Come sopra.
- **Server:** La controparte di **User-Agent**.
- **Set-Cookie:** Comunica al client che esso dovrebbe memorizzare un cookie con un certo nome, valore, e facoltativamente scadenza, dominio, percorso e flag di sicurezza. Esempio: **Set-Cookie: score=127**. Una volta che **Set-Cookie** viene ricevuto ed accettato, il client invierà al server il cookie ad ogni richiesta eseguita.
- **Content-Length:** Specifica la grandezza in byte del response body. In caso "apparisse" dal lato del richiedente, dobbiamo fare attenzione a specificare la lunghezza giusta in caso volessimo modificare i nostri payload.

Negli esempi mostrati precedentemente potete vedere come questi header vengono utilizzati in una comunicazione reale tra un web browser e un sito web statico.

Generalmente, quando un header inizia per **X-**, è custom. È utile notare come il funzionamento di HTTP sia solo una convenzione, ed il server può decidere di implementare qualsiasi metodo e qualsiasi header (custom headers e methods). Questi elementi sono di nostro interesse, essendo implementati direttamente dal gestore del sito e quindi più facilmente soggetti ad errori di implementazione. Inoltre, nulla impedisce al programmatore di usare una GET per modificare dati, o una POST per fornire informazioni, nonostante questo sia ovviamente sconsigliato. Lo stesso vale per gli elementi mostrati successivamente in questo capitolo.

#### Status codes:

- **1xx**: Risposte informative
- **2xx**: Successo
- **3xx**: Reindirizzamento
- **4xx**: Errore del client (tipicamente richieste sbagliate)
- **5xx**: Errore del server (errori di un programma ed eccezioni non gestite)

**Riassumendo:** Possiamo pensare all'HTTP request come a una lettera che spediamo tramite posta. Nella riga della richiesta, noi come mittenti specifichiamo cosa vogliamo sia fatto e dove vogliamo sia fatto, come scrivessimo l'indirizzo e il tipo di servizio desiderato su una busta. Gli header della richiesta contengono informazioni su di noi e le nostre preferenze, simili a scrivere il nostro nome e il nostro indirizzo sul retro della busta. Se chi offre il servizio ha bisogno di un materiale o di un oggetto da utilizzare per soddisfare la nostra richiesta, possiamo includerlo nel body message, proprio come inviare un pacco insieme alla busta.

Il server, simile al destinatario della nostra lettera, riceve la richiesta, cerca di soddisfarla e ci invia una lettera di risposta. Nella riga dello stato della risposta, capiamo se tutto è andato bene o se c'è stato un problema, proprio come leggere l'indicazione di consegna sulla nostra busta postale. Negli header della risposta, otteniamo informazioni su chi ha eseguito il lavoro e come vorrebbe che ci comportassimo con il risultato. Infine, nel body message della risposta, riceviamo il prodotto richiesto, come se insieme alla busta ci venisse spedito un pacco contenente ciò che avevamo chiesto.

#### HTTP Cookies

I cookie vengono spesso arricchiti da attributi, ed i principali sono: - **Expires**: Specifica il tempo di scadenza in secondi per il cookie. Se non specificato, il cookie viene eliminato al termine della sessione (session cookie). - **Secure**: I cookie con questa flag vengono inviati solo in richieste HTTPS (HTTP crittografato). - **HttpOnly**: JavaScript non può accedere al cookie. - **Domain**: Definisce il dominio per il quale il cookie è valido. - **Path**: Stessa cosa ma col percorso. - **Same-Site**: Specifica se il cookie può venire incluso in richieste che coinvolgono siti terzi. **SameSite=Strict** indica che il browser si rifiuterà di condividere il

cookie con siti web diversi da quello che ci ha “detto” di settare il cookie. È una protezione che può scoraggiare attacchi CSRF.

Un cookie (nel browser: F12 -> Application -> Cookies):

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
__Secure-1PSIDCC	ABTWbQFjb4ZKHFI4adWEQnQQTbtk	.google.com	/	2025-02-18T16:08:40.000Z	44	✓	✓	

Figure 7: Esempio di cookie

## HTTP Authentication

Secondo gli standard del protocollo HTTP, la struttura della comunicazione tra un client che richiede una risorsa protetta e il server consiste nei seguenti step: - Il client richiede la risorsa - Il server risponde con lo status code **401 Unauthorized** (non autorizzato), specificando tramite l'header **WWW-Authenticate** il tipo di autenticazione richiesto. In questa fase possono essere inviate al client diverse informazioni, a seconda del metodo di autenticazione richiesto. - Il client deve rispondere con l'header **Authorization** contenente le credenziali richieste. - Il server risponde **200 OK** o **403 Forbidden** (accesso vietato).

**401 Unauthorized** = “non so chi sei”, **403 Forbidden** = “non sei un utente che ha accesso alla risorsa”.

### Principali tipi di autenticazione:

- **Basic Authentication:** `username:password` vengono inviati encodati in `base64`. L'encoding non offre nessun layer di sicurezza aggiuntivo, di conseguenza un'autenticazione **Basic** in HTTP è completamente insicura, come mandare le informazioni in chiaro.
- **Digest Authentication:** Offusca username e password usando anche altri parametri come `realm` e `nonce`
- **Bearer Authentication:** Usata soprattutto in contesti basati su OAuth2. Di fatto, invece di fornire le credenziali al server che richiede l'autenticazione, ci autenticiamo presso un altro server del quale il server iniziale si fida. Questo è possibile perchè il server autenticante fornisce all'utente un token da usare come “cartellino d'ingresso” quando passiamo all'ultima fase del processo di autenticazione. Spesso i token sono generati come JWT.

*Nota: I tipi di autenticazione potrebbero non essere chiari fin da subito, e non sono necessariamente spendibili in attacchi utili al contesto di CyberChallenge, ma l'utilizzo delle Bearer authentication è in continua crescita ed è un argomento che ritengo particolarmente importante. Ne consiglio caldamente l'approfondimento autonomo.*

**Conclusione del capitolo:**

Quando difendiamo o attacchiamo un servizio, è utile ricordare che i cookies, gli header, il body content e il metodo della richiesta possono venire modificati dal client come esso vuole. Esistono strumenti (come BurpSuite) che permettono di modificare facilmente tutte le informazioni possibili che il server è in grado di ricevere. Fidarsi di ciò che viene inviato dal client significa accettare di gestire un servizio vulnerabile. Uno sviluppatore deve fare in modo di limitare per quanto possibile le funzionalità che richiedano una fiducia dell'utente. La natura stateless di HTTP costringe gli sviluppatori ad usare i cookie per l'autenticazione (immaginate di dover loggare ogni volta che cambiate reel), mettendoli in difficoltà e aprendo la possibilità a vari tipi di attacchi cross-site che vedremo più avanti.

*Challenge: Prime 6 di web security a partire da questa*