Attacchi Web
Introduzione alla sicurezza nelle applicazioni Web

Davide Marrone
davide@security.dico.unimi.it

Università degli Studi di Milano
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione

15 gennaio 2007

Sommario

Cos'è il Web

- È un sistema nato per distribuire su internet documenti o elementi multimediali collegati tra di loro
- Negli anni si è trasformato in un'infrastruttura complessa sulla quale realizzare applicazioni distribuite
- Questa trasformazione ha introdotto nuove vulnerabilità
- È basato sul paradigma client/server
- Sono presenti in rete milioni server, che ospitano applicazioni web, in attesa di richieste da parte di client
- I client comunicano con i server attraverso il protocollo `HTTP`

Richiesta di una pagina

Client

- Per richiedere una pagina l'utente inserisce il link nel browser
- Il browser effettua una query al DNS per ottenere l'indirizzo ip del server associato al dominio presente nel link
- Il browser crea una socket, si collega alla porta 80 dell'ip ottenuto e invia una richiesta HTTP

Server

- È in ascolto sulla porta 80 TCP in attesa delle richieste dei client
- Riceve le richieste, le processa e ritorna una risposta HTTP

Client

- Riceve la risposta e la elabora, nella maggior parte dei casi il payload del protocollo HTTP è un documento HTML

HyperText Transfer Protocol

HyperText Transfer Protocol

- È un protocollo di livello applicazione usato per trasferire dati tra un client ed un web server
- La versione 1.0 è definita nel RFC 1945
- La versione 1.1 è definita nel RFC 2616
- Negli RFC si presume che sia utilizzato sopra ad un protocollo affidabile
- Su internet viene incapsulato all'interno di connessioni `TCP`, di default si utilizza la porta 80
- Lo scopo originale del protocollo era quello di ottenere documenti ed elementi multimediali, oggi viene utilizzato anche per trasportare altre informazioni (es. `SOAP`)

Richieste HTTP

Una richiesta è composta da:

1. **Request line** (GET /index.html HTTP/1.1)
2. Una serie di **header** (opzionali)
   (User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1))
3. Una linea vuota
4. Il corpo del messaggio (opzionale)

- La `request line` e gli `headers` devono avere, in fondo alla riga, un `CRLF` (carriage return seguito da un line feed: "\r\n")
- La linea vuota deve essere fatta da solo da `CRLF` senza spazi
- In realtà le richieste vengono accettate anche se c'è solo un `LF`
- Con la versione 1.1 tutti gli `header`, tranne `Host`, sono opzionali

Request line e metodi HTTP

La Request line è composta da:

1. Metodo (es. GET, HEAD, POST, ...)
2. Risorsa (es. /index.html)
3. Versione del protocollo (es. HTTP/1.1)

Metodi (GET /index.html HTTP/1.1)

GET serve per ottenere la risorsa specificata

HEAD simile alla GET, in questo caso però, la risorsa non viene restituita ma vengono restituiti solamente le informazioni relative al protocollo

POST indica al server che nella richiesta, oltre agli header HTTP, sono presenti dei dati nel body, possono essere:

- Dati da inserire in un forum, newsgroup, mailing list, ...
- Dati provenienti da un form di un'altra pagina HTML
- Input che devono essere inseriti in un database

Nella versione 1.1 ne sono stati aggiunti altri (OPTIONS, PUT, DELETE, TRACE, CONNECT)

Risorsa e Versione del protocollo

Risorsa (GET /index.html HTTP/1.1)

- La risorsa può essere specificata inserendo una URI assoluta oppure il path assoluto
- Si può inserire la URI assoluta solo quando si effettua una richiesta attraverso un proxy
  (es. GET http://security.dico.unimi.it/index.html HTTP/1.1)
- Si inserisce il path assoluto della URI quando si effettuala la richiesta direttamente al server
  (es. GET /index.html HTTP/1.1)

Versione del protocollo (GET /index.html HTTP/1.1)

- 1.1 indica il major e minor number della versione del protocollo
- Nelle richieste della versione 1.1 è obbligatorio inserire l'header **Host**
- Normalmente viene utilizzata la versione 1.1

I principali header

Request Header

Authorization si possono specificare delle credenziali: Basic seguito da "username:password" codificato in base64
(es. davide:davide ⇔ ZGF2aWRlOmRhdmlkZQ==)

If-Modified-Since viene specificata una data, se la risorsa non è stata modificata da quella data il server non la restituisce e il client usa la sua copia locale

Referer indica la pagina di provenienza

User-Agent indica l'agente con cui è stata effettuata la richiesta

Entity Header

Content-Length indica la lunghezza del payload contenuto nella richiesta

Content-Type indica il contenuto del body
(es. application/x-www-form-urlencoded)

Una richiesta HTTP (GET)

Esempio GET

```
GET /test.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Iceweasel/2.1
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,*/*;q=0.5
Accept-Language: it,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost/
If-Modified-Since: Fri, 05 Jan 2007 14:30:45 GMT
If-None-Match: "7f0f1-7-ecdaaf40"
```

Una richiesta HTTP (POST)

Esempio POST

```
POST /textlegth.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Iceweasel/2.1
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,*/*;q=0.5
Accept-Language: it,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost/
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

testo=prova
```

Risposte HTTP

Una risposta è composta da:

1. **Status-Line** (es. HTTP/1.1 200 OK)
2. Una serie di **header** (opzionali) (es. Server: Apache/2.2.3)
3. Una linea vuota
4. Il corpo del messaggio (opzionale)

Valgono le stesse regole presenti nella richiesta

- La status line e gli headers devono avere alla fine un CRLF
- La linea vuota deve essere fatta solo da CRLF senza spazi

La Status line è composta da:

1. Versione del protocollo (es. HTTP/1.1)
2. Status code: codice numerico che indica il risultato (es. 200, 404, ...)
3. Testo associato al codice numerico (es. OK)

Codici di risposta

Status code

Il codice è composto da tre numeri, il primo indica la classe della risposta, gli altri due indicano in dettaglio il significato.
Ci sono 5 classi di risposta:

- 1xx Informazione - La richiesta è stata ricevuta parzialmente, il client deve completarla (introdotti nella versione 1.1)
- 2xx Successo - L'azione è stata ricevuta, capita, e accettata con successo
- 3xx Redirezione - Sono necessarie altre azioni per completare la richiesta
- 4xx Client Error - La richiesta ha una sintassi errata oppure non può essere soddisfatta
- 5xx Server Error - Il server non è in grado di rispondere ad una richiesta valida

Codici di risposta

Esempi

200 OK

201 Created

202 Accepted

301 Moved Permanently

307 Temporary Redirect

400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

500 Internal Server Error

503 Service Unavailable

I principali header

Response Header

Server informazioni generali sul server web

Location utilizzato nei redirect, indica la locazione della risorsa

Last-Modified usato per la cache, dice quando è stata modificata l'ultima volta la risorsa

Content-Length indica la lunghezza del payload `HTTP`

Content-Type indica il contenuto del payload `HTTP`

Una risposta HTTP

Esempio di richiesta/risposta

```
GET /test.html HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Date: Fri, 05 Jan 2007 14:55:09 GMT
Server: Apache/2.2.3 (Debian) PHP/4.4.4-8
Last-Modified: Fri, 05 Jan 2007 14:30:45 GMT
ETag: "7f0f1-7-ecdaaf40"
Accept-Ranges: bytes
Content-Length: 5
Content-Type: text/html; charset=UTF-8

prova
```

Sessioni HTTP

- HTTP è un protocollo stateless, non esiste il concetto di sessione, ogni richiesta è indipendente dalle precedenti
- Nessun problema per i siti statici ma insufficiente per applicazioni Web dinamiche
- La maggior parte di applicazioni Web ha la necessità di tenere traccia delle informazioni relative ad ogni utente
- Le sessioni vengono implementate dall'applicazione Web
- I dati relativi alle sessioni devono inevitabilmente essere passati ad ogni richiesta dal client al server
- Le informazioni sono inserite all'interno del protocollo HTTP, possono essere inserite in tre punti:
  1. In particolari header HTTP (Cookie)
  2. All'interno delle URL
  3. Nel payload delle richieste HTTP

Cookies

Funzionamento

- Sono dati creati dal server e memorizzati nel client
- Vengono scambiati tra client e server utilizzando degli header particolari aggiunti agli header `HTTP` standard
- Per creare un cookie sul client il server aggiunge un header del tipo:
  `Set-Cookie: prova=test<CRLF>`
- Il client memorizza localmente l'informazione e nelle richieste successive aggiunge il seguente header:
  `Cookie: prova=test<CRLF>`
- Sono stati standardizzati dal
  `RFC 2109: HTTP State Management Mechanism`
- Un esempio tipico di utilizzo è la registrazione di una preferenza dell'utente, es. il numero di risultati da visualizzare per una ricerca

Cookies - struttura

Come sono fatti

nome=valore	dati generici (unico campo obbligatorio)
expires	data di scadenza
path	percorso per il quale il cookie è valido
domain	dominio per il quale il cookie è valido (es. .foo.it)
secure	flag che indica se il cookie deve essere trasmesso solo attraverso un canale sicuro
HttpOnly	flag **non** standard introdotto da Microsoft per contrastare i XSS, funziona solo con Internet Explorer >= 6.0

Realizzazione di una sessione

Realizzazione di una sessione

Ci sono due possibilità per realizzare una sessione HTTP:

1. Il programmatore inserisce nell'HTTP i dati che ritiene necessari (obsoleto e probabilmente insicuro)
2. Il programmatore utilizza il meccanismo implementato dal linguaggio di programmazione

Cookies di sessione

- La maggior parte di linguaggi di programmazione (lato server) permettono la creazione di una sessione
- Tutti i dati relativi al client sono memorizzati sul server
- Al client viene passato un unico cookie che contiene l'id di sessione
- ad ogni richiesta il client spedisce i cookie al server (Cookie: prova=ciao; PHPSESSID=da1dd139f08c50b4b1825f3b5da2b6fe)
- Il server, attraverso l'id, recupera le informazioni per quel client

Sicurezza sulle sessioni

- Sono un elemento critico, possono essere degli autenticatori
- Se implementate in maniera errata possono permettere l'elusione dell'autenticazione
- Attacchi possibili agli id di sessione:
  - Intercettazione $\longrightarrow$ `SSL/TLS`
  - Predizione $\longrightarrow$ strong pseudonumber
  - Brute force $\longrightarrow$ lunghezza id
  - Session fixation $\longrightarrow$ controllo `IP`, `Referer`; rigenerazione id; ...
- Devono essere valide per un periodo di tempo limitato

Tecnologie per la programmazione Web

Client side

Si basano sull'esecuzione di codice sulla macchina client

1. Sono applicazioni indipendenti
2. Utilizzano moduli integrati nel browser

Server side

Si basano sull'esecuzione di codice sulla macchina server

1. Sono dei moduli del server web
2. Sono dei processi indipendenti che comunicano con il server web, questi processi posso essere residenti anche su macchine diverse

Tecnologie presenti sul client

- Java applets, ActiveX controls
- Flash (ActionScript), Shockwave
- JavaScript/JScript/EcmaScript, VBScript
- AJAX

Applicazioni indipendenti

Java Applets

- Sono programmi Java compilati
  - Vengono scaricati dal browser
  - Sono eseguiti all'interno di una pagina web
- Hanno accesso alle risorse in base ai permessi del Java Security Manager

ActiveX Controls

- Versione Microsoft delle applets
- Sono dipendenti dal sistema operativo e dal browser
- Se eseguiti hanno pieno accesso al sistema operativo del client

Moduli integrati nel browser

JavaScript/JScript/EcmaScript - VBScript

I linguaggi di scripting sono utilizzati per realizzare comportamenti dinamici nelle pagine web

- **JavaScript** è stato il primo linguaggio introdotto da Netscape
- **JScript** è la versione di Microsoft
- **EcmaScript** è la standardizzazione di JavaScript
- **VBScript** è un linguaggio basato su Microsoft Visual Basic, funziona solo con Internet Explorer

Sicurezza scripting

- Da JavaScript sono accessibili i cookie creati per il dominio da cui è stato generato lo script
- Il flag `HttpOnply` ha proprio lo scopo di impedire l'accesso ai cookie
- Alcuni attacchi si basano in parte o completamente su JavaScript (XSS, XSRF)
- La soluzione migliore per la sicurezza sarebbe quella di disabilitare l'esecuzione di script lato client
- Moltissimi siti non funzionano senza JavaScript e stanno nascendo nuove applicazioni Web basate totalmente su JavaScript (AJAX)

Tecnologie lato client

Sicurezza

- Non dovrebbero essere utilizzate per effettuare nessun tipo di autenticazione basato esclusivamente sul client
- Il codice eseguito sul client può essere letto e modificato
- Non vanno lasciati commenti con informazioni importanti
- Non vanno usati i campi hidden dell'`HTML` per memorizzare informazioni su cui fare affidamento
- I dati provenienti dal client vanno sempre considerati insicuri
- I controlli lato client possono solo migliorare l'usabilità dell'applicazione e il carico del server
- I controlli di sicurezza vanno sempre implementati lato server

Tecnologie presenti sul server

- CGI (common gateway interface)
- PHP (Hypertext Preprocessor)
- PERL, Python, Ruby
- ASP (Active server pages)
- ASP .Net
- JSP (Java server pages)
- Servlet
- ColdFusion

Passaggio di parametri GET

- L'utente passa i dati all'applicazione Web attraverso form HTML o con tecnologie lato client
- Tutto si deve però tradurre in una richiesta HTTP

```
<form action="test.php" method="get">
<input type="text" name="var1" />
<input type="hidden" name="var2" value="b" />
<input type="submit" value="invia" />
</form>
```

```
<a href="test.php?var1=a&var2=b">prova</a>
```

```
GET /test.php?var1=a&var2=b HTTP/1.1
Host: localhost
...
...
```

Passaggio di parametri POST

```
<form action="test.php" method="post">
<input type="text" name="var1" />
<input type="text" name="var2" />
<input type="submit" value="invia" />
</form>
```

```
POST /test.php HTTP/1.1
Host: localhost
...
...
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

var1=a&var2=b
```

```
<form action="test.php?var3=c&var4=d"
method="post">
<input type="text" name="var1" />
<input type="text" name="var2" />
<input type="submit" value="invia" />
</form>
```

```
POST /test.php?var3=c&var4=d HTTP/1.1
Host: localhost
...
...
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

var1=a&var2=b
```

Analisi di traffico HTTP

Analisi di traffico HTTP

- Il protocollo HTTP è incapsulato all'interno del protocollo TCP, di default si usa la porta 80
- Tutto il traffico HTTP passa in chiaro
- Si può analizzare tutto il traffico tra client e server con i soliti strumenti per l'analisi del traffico di rete (ngrep, tcpdump, tshark, wireshark, ...)
- Per windows esistono degli sniffer dedicati solamente al protocollo HTTP (httpanalyzer, HTTPLook, HttpSpy, ...)
- Con i tradizionali sniffer non è possibile analizzare il traffico HTTP che viene incapsulato in una connessione cifrata con TLS/SSL

Analisi, modifica e creazione di traffico HTTP

Analisi e modifica di traffico HTTPS

Per sniffare traffico cifrato con TLS/SSL ci sono due possibili alternative. Entrambe, oltre alla possibilità di analizzare il traffico, permettono anche di modificarlo

1. Bisogna utilizzare delle estensioni del browser che hanno accesso al traffico in chiaro e possono intercettare le richieste e le risposte (Per `Firefox` c'è `Tamper Data`)
2. Bisogna utilizzare un proxy HTTP

Tools per la creazione di richieste HTTP

- Browser tradizionali (Firefox, Internet Explorer, wget, ...)
- netcat
- curl

Proxy HTTP

- Sono dei programmi che permettono l'analisi del traffico HTTPS
- Permettono di modificare il traffico HTTP e HTTPS scambiato tra client e server
- Sono totalmente indipendenti dall'applicazione, l'unico requisito è la possibilità di settare il proxy nell'applicazione da analizzare
- Se si utilizzano per intercettare traffico HTTPS il browser emetterà un warning avvisando che il certificato SSL non è valido

I proxy HTTP più famosi

- WebScarab - http://www.owasp.org/
- Burp - http://www.portswigger.net/proxy/
- Paros - http://www.parosproxy.org/

Bibliografia

Bibliografia

- RFC HTTP

  http://tools.ietf.org/html/rfc1945 (HTTP 1.0)
  http://tools.ietf.org/html/rfc2616 (HTTP 1.1)
  http://tools.ietf.org/html/rfc2109 (HTTP State Management Mechanism)

- Altro

  http://wp.netscape.com/newsref/std/cookie_spec.html (Cookies)
  http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp (HttpOnly)
  https://addons.mozilla.org/firefox/966/ (Tamper Data)