***RELAZIONE LABORATORIO IncApache***

Lo scopo del laboratorio Incapache era di implementare un web server in grado di gestire le richieste HTTP e rispondere rispettando le specifiche del protocollo.

Per la versione 2.3 inoltre il programma prevede l'utilizzo di 8 thread in ascolto e 24 di risposta.

* ***Main()***

Il processo inizia subito controllando che l'effective uid sia Root mentre il real uid sia impostato su un'utente diverso dall'amministratore.

Successivamente crea attraverso due pipe un canale bidirezionale con il file, ottiene il percorso per la directory corrente e crea un child process che determina ,di volta in volta, il tipo di file passato.

Il processo principale chiama in seguito la funzione run\_webserver() che provvederà ad impostare come directory root la cartella passata attraverso chroot(), ad inizializzare un socket in ascolto e a creare gli 8 processi in ascolto.

Infine dopo avere rilasciato i privilegi di root, utilizza la pthread\_join() per forzare la partenza dei processi in ascolto.

* ***Manage\_http\_requests()***

Questa funzione serve per interpretare le richieste http raccolte ed in base alle specifiche del protocollo rispondere con i diversi codici utilizzando la funzione SEND\_RESPONSE().

Quest'ultima può essere l'alias della funzione send\_response(), se la versione utilizzata è la 2.2, altiemnti corrisponde alla funzione send\_response\_thread().

Per la divisione in token del campo "if-modified-since" abbiamo utilizzato la funzione thread safe strtok\_r.

* ***Send\_response()***

In questa funzione viene calcolata l'ora in cui viene ricevuta la richiesta utilizzando la funzione gmtime\_r() (che utilizza un suo buffer interno per risultare thread\_safe) e vengono inizializzate le informazioni per l'header di risposta relativo al file richiesto richiamando stat() e get\_mime\_type().

Per concludere la funzione invia l'intestazione del file richiesto più quest'ultimo se il metodo di richiesta GET.

* ***Get\_mime\_type()***

La funzione ha l'obiettivo di restituire il tipo di file passato.

Per evitare collisioni tra thread abbiamo utilizzato un lock che garantisce un accesso alla volta sul file.

Per testare il programma abbiamo provato delle richieste legittime utilizzando i metodi HEAD, GET e CONDITIONAL GET supportati dal server.

**HEAD / HTTP/1.0** ha restituito come previsto 200 OK

**HEAD /images/uncadunca.jpg http/1.1** ha restituito 200 OK

**GET /images/uncadunca.jpg HTTP/1.1** ha restituito 200 OK

**GET /images/uncadunca.jpg HTTP/1.1**

**If-Modified-Since: Mon, 26 Nov 2014 11:27:51 GMT** ha restituito (2 casi)

1)304 Not Modified

2)dopo il comando touch della risorsa richiesta 200 OK

Abbiamo successivamente eseguito numerose prove con richieste contenenti (pensiamo) tutti gli errori possibili che un utente potrebbe inavvertitamente commettere

**HEAAD / HTTP/1.0** ha restituito 501: Method Not Implemented

HEAD \ HTTP/1.0 400 Bad Request

**HEAD /aaa.html HTTP/1.0** ha restituito 404 Not Found

**HEAD /images/uncadunca.jpg HTTTP/1.0** 400 Bad Request

**GET (../) HTTP/1.1** ha restituito 404 Bad request

**HEAD /../main.c HTTP/1.0** 404 Not Found

Abbiamo poi verificato che il server rispondesse alla richiesta nonostante errori presenti nella option line

**GET /images/uncadunca.jpg HTTP/1.1**

**If-Modifd-Since: Mon, 26 Nov 2014 11:27:51 GMT** ha restituito

Abbiamo infine testato il server con i browser Firefox,Chrome,Edge e controllato la gestione su possibili buffer overflow utilizzando stringhe di grandi dimensioni.

Test per buffer overflow:

**GET /images/../images/../(il percorso ripetuto molte volte)/images/../images/../images/../images/../images/../images/../images/uncadunca.jpg HTTP/1.0**

ha restituito Bad Request

**HEE(molti altri caratteri )A / HTTP/1.1**

ha restituito Bad Request

**stessa cosa per la stringa HTTP/1.1**

**GET /images/uncadunca.jpg HTTP/1.1**

**If-Moo(...)o: Mon, 25 Nov 2017 11:11:11 GMT** ha restituito invece la risorsa ignorando la richiesta option line come il pre-compilato.