**UNIVERSITA' DEGLI STUDI DI FIRENZE**
FACOLTA' DI INGEGNERIA INFORMATICA

Seminario

# CONCETTI DI SICUREZZA IN PHP

**Dott. Ing. Stefano Di Paola**

# Sicurezza - Concetti Generali

**Sicurezza:** *"Stato oggettivo sul quale l'individuo ripone un grado atteso di tranquillità"*

- La mancanza delle misure necessarie a creare un grado atteso di tranquillità porta ad un *falso senso di sicurezza*.
- Quali sono queste misure necessarie?
  - Bilanciare *Fiducia e Diffidenza* verso il prossimo.
  - Bilanciare *Umiltà e Superbia.*
  - Avere *Consapevolezza e Conoscenza.*
- La fonte principale di insicurezza è sempre *l'errore e l'ingenuità umana!*.

# Sicurezza - Concetti Generali

- La fiducia nel prossimo è spesso necessaria.
  - Chiedere una informazione a uno sconosciuto..
  - Fidarsi del responso di un elettricista che ci ha aggiustato l'impianto...
  - Non si ha sempre il controllo su come le informazioni e i servizi vengono elargiti...
  - ...e se qualcuno ci chiede di entrare in casa a vedere il contatore?
- Quando siamo noi ad avere tale controllo allora entriamo in gioco.

# Sicurezza - Concetti Generali

- Quando si è sistemista, progettista o sviluppatore software il controllo è, almeno in parte, nostra esclusiva responsabilità.
- Se si lascia l'utente ad assumere tale ruolo le conseguenze possono essere nefaste!!
  - Perdita dell'integrità dei dati
  - Corruzione dei dati
  - Diffusione non autorizzata dei dati
  - Utilizzo non autorizzato di risorse.
- ...allora non si può più prevedere il risultato di una qualunque azione del sistema.

# Sicurezza Informatica - Concetti Generali

**Sicurezza Informatica:** *"L'insieme delle precauzioni e delle misure che si possono e si devono prendere per far si che l'accesso ai dati, e più in generale alle risorse informatiche di una particolare macchina o rete sia consentito solo a chi ne ha effettiva autorizzazione"*

# Sicurezza Informatica - Concetti Generali

**Internet è**

- Una rete accessibile a tutti.
- Un mondo con le sue regole.
- Piena di utenti inconsapevoli.
- Piena di gente che cerca di:
  - farsi un nome cercando di trovare vulnerabilità
  - distruggere il contenuto di siti e dati
  - risolvere la proria giornata
  - fare soldi truffando.
- Non importa se si ha un sito piccolo o grande.
- Non importa se un programma è utilizzato da due o mille utenti.
- Importa se sei in rete o no.
- Importa se sei accessibile o no.

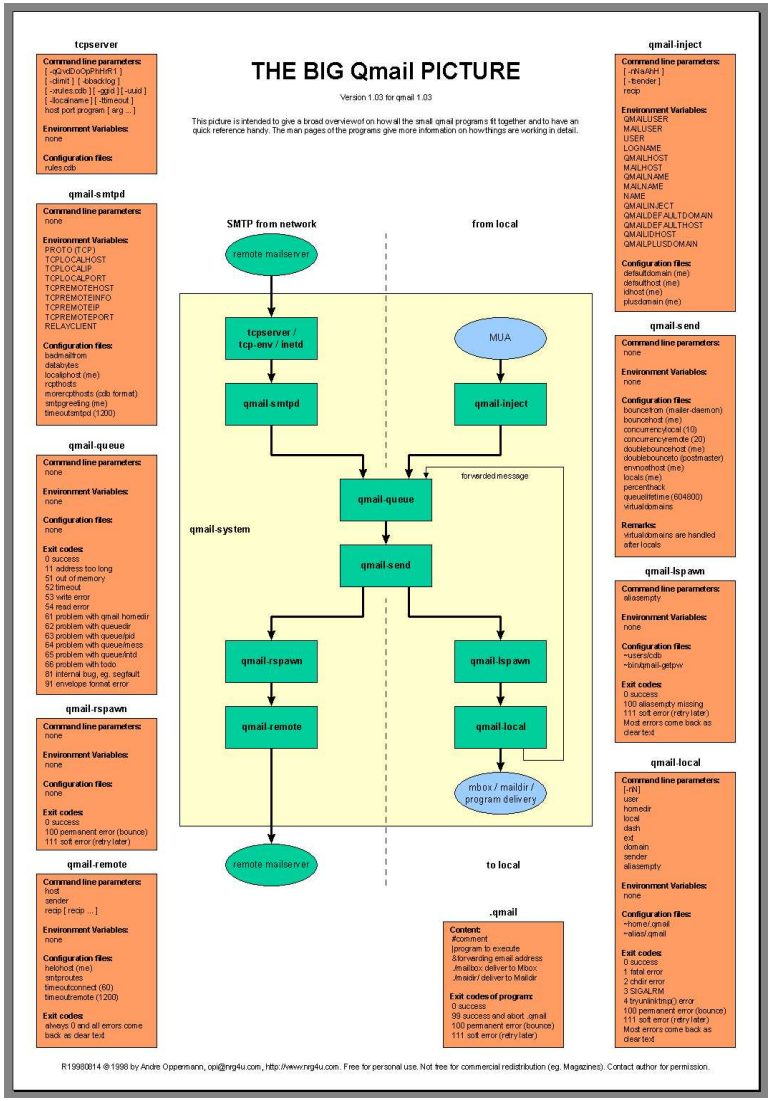
# Sicurezza Informatica - Concetti Generali

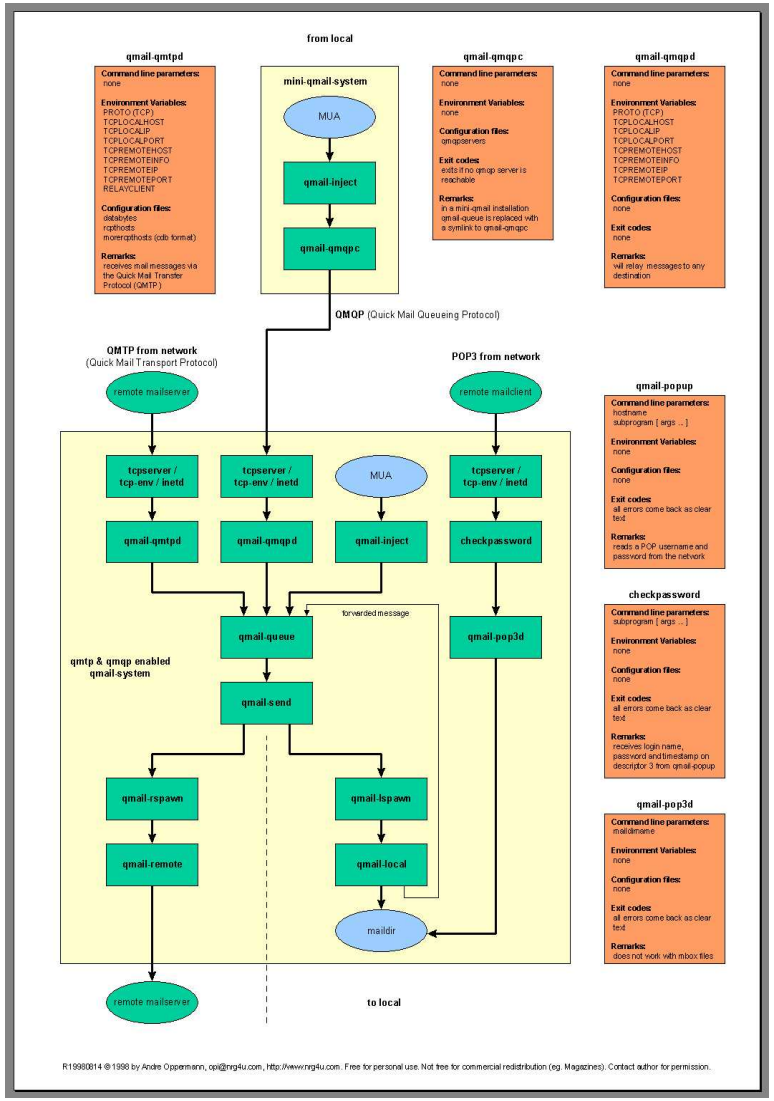
*L'ingegneria del software insegna che:*

- Realizzare un software senza errori è alquanto difficile.
- È ancora peggio quando si trascurano o si ignorano gli aspetti elementari di sicurezza.
- Non esiste *un sistema invulnerabile.*
- Esiste il *principio del minimo rischio.*

# Sicurezza Informatica - Un esempio

Qmail - Un programma concepito con la sicurezza in mente.



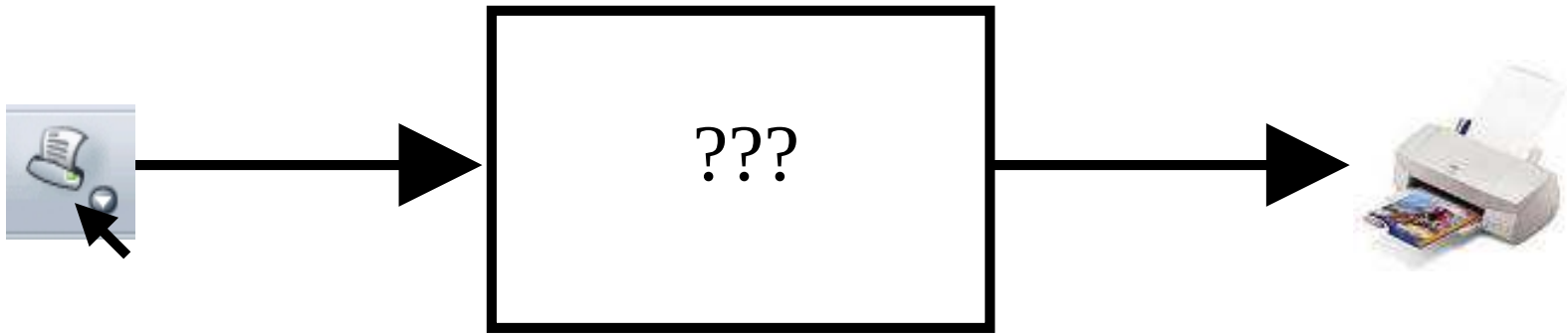
# Input, Output e Black Box

Il Software è di fatto modellabile tramite lo schema seguente:

# Input, Output e Black Box

**Dal punto di vista dell'utente generico**

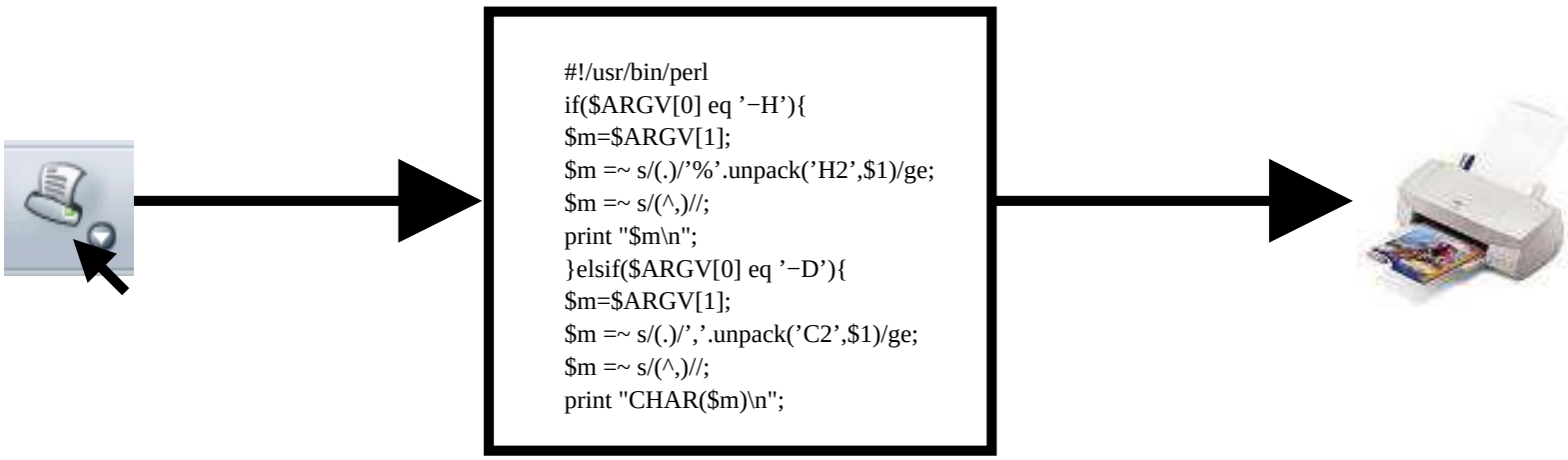
- il software si comporta come una black box:
  Per ogni insieme di dati che l'utente immette nel software egli ottiene un insieme di dati che il software restituisce.



# Input, Output e Black Box

**Dal punto di vista dell'utente esperto**:

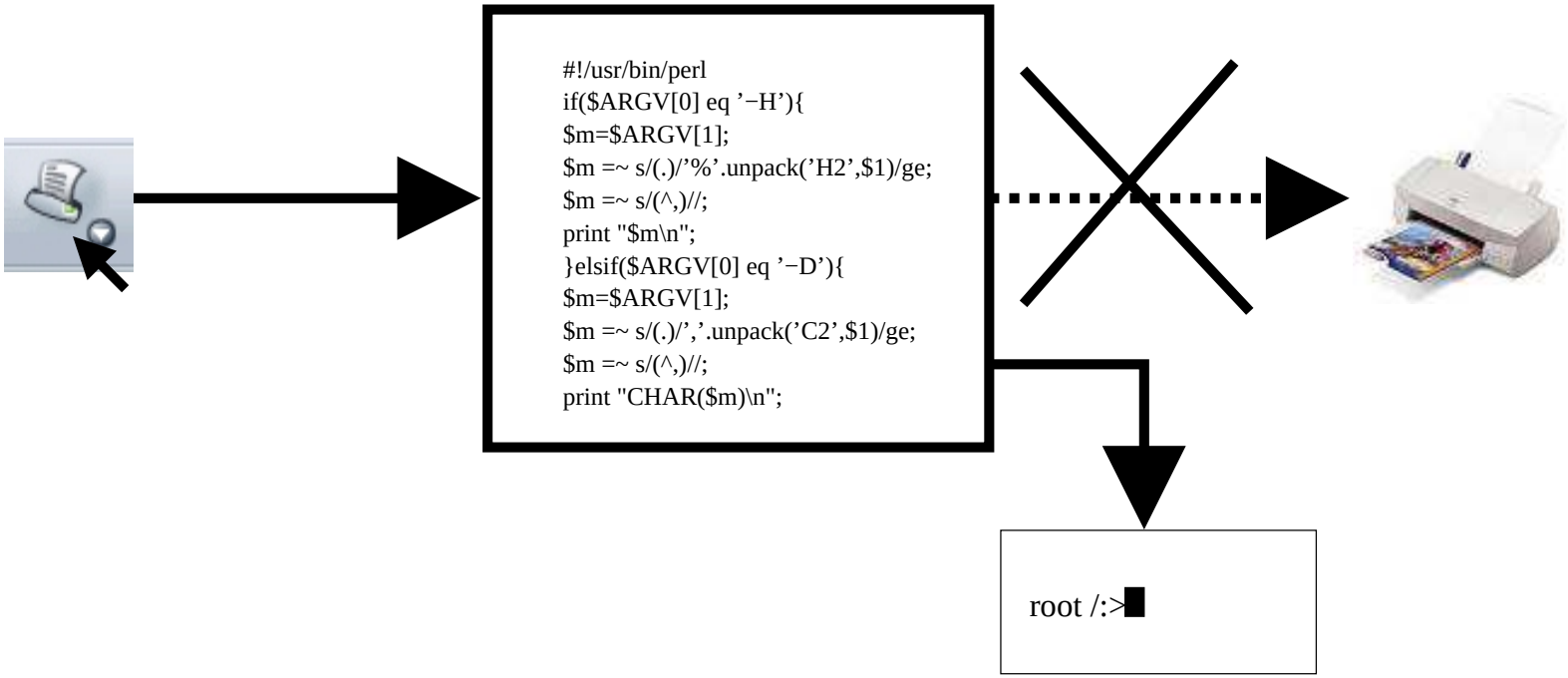
- Per una tipologia di input il software trasforma, come una qualunque funzione matematica per quanto complessa essa sia, un output.



# Input, Output e Black Box

**Dal punto di vista dell'hacker**:

- Attraverso la manipolazione dell'input e dell'output egli cerca di mettere in crisi il software per ottenere un comportamento non originariamente previsto.



# Input, Output e Black Box

**Concetti Errati**:

- *Security Through Obscurity.*
  Software proprietario $\Rightarrow$ software sicuro.
- *Sorgente aperto = Insicurezza:*
  Se posso analizzare un software $\Rightarrow$ lo posso attaccare più facilmente.

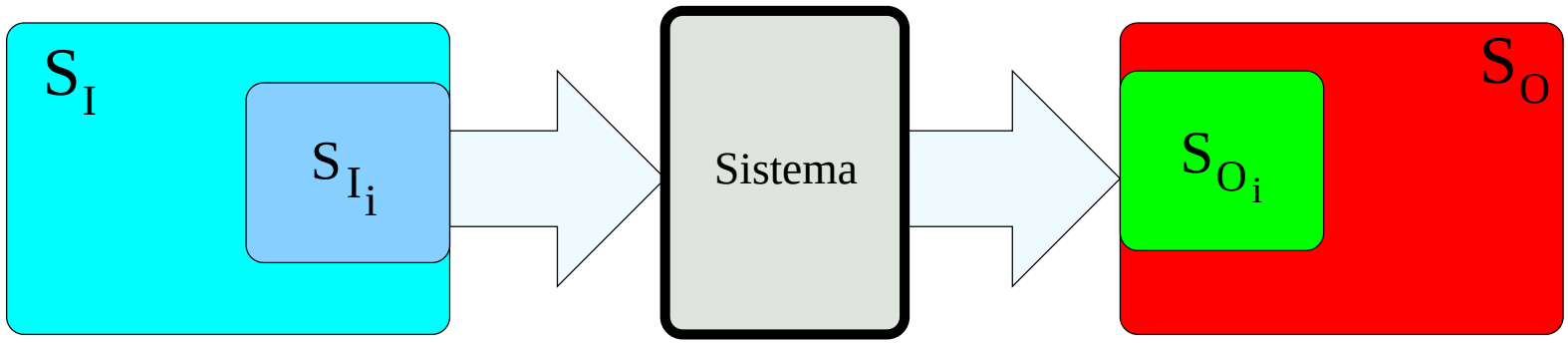
**Concetti Giusti**:

- Conosco in profondità l'ambiente di lavoro $\Rightarrow$ Conosco la trasformazione Input - Output a basso livello.
- Conosco il sottoinsieme di dati in Input $\Rightarrow$ conosco il sottoinsieme di dati in Output

# Sottoinsiemi di Input e Output

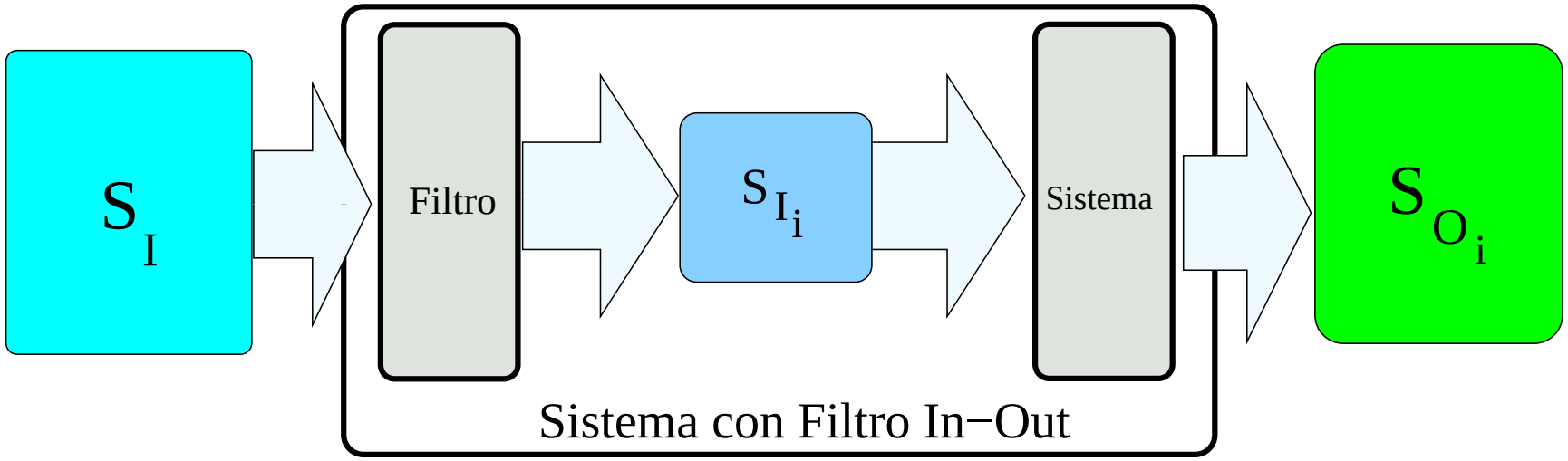
Definizioni:
**Spazio degli Input** $S_I$: L'insieme di tutti gli *ingressi* che possono essere passati ad un sistema.
**Sottospazio degli Input** $S_{I_i}$: L'insieme di tutti gli *ingressi* che possono essere passati ad un sistema ottenendo un atteso comportamento da parte del sistema stesso.

**Spazio degli Output** $S_O$: L'insieme di tutti i *risultati* che un sistema può produrre.
**Sottospazio degli Output** $S_{O_i} \subseteq S_O$: L'insieme di tutti i *risultati* che si desidera che un sistema produca.



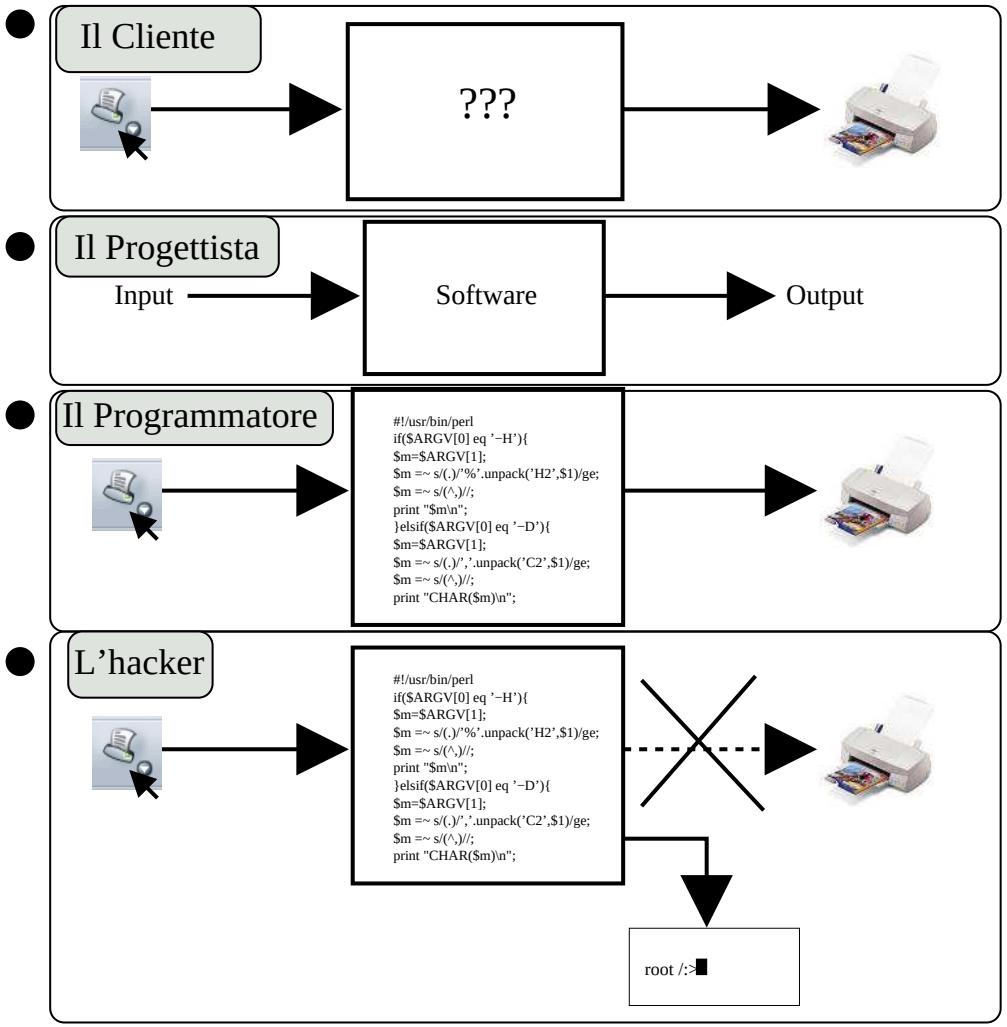
# Sistema con Filtro In-Out

La principale metodologia di difesa consiste nel generare *sempre* da $S_I \Rightarrow S_{I_i} \subseteq S_I \Rightarrow S_{O_i}$ tramite un sistema di filtraggio.



# Sicurezza e Punti di Vista

Esistono più punti di vista:




In qualunque ruolo si giochi il punto di vista della sicurezza deve essere unico.

In qualunque ruolo si giochi si deve conoscere il punto di vista degli altri giocatori.

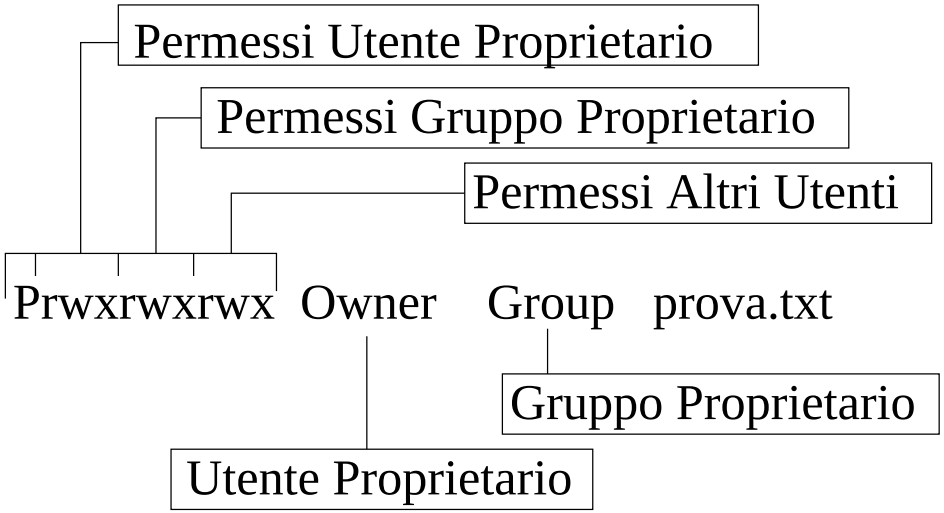
# *Concetti Propedeutici*

# S.O. Avanzati - Permessi e Proprietari

Vi sono due principali tipologie di S.O.

- Con politiche di gestione degli utenti (Unix, Windows 2000, XP).
- Senza politiche di gestione degli utenti (Windows 9x/ME).

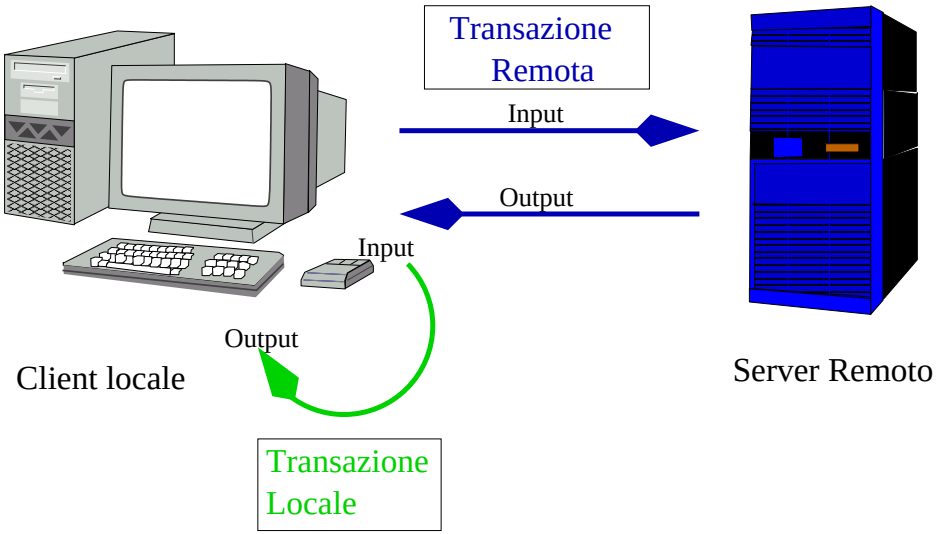
La gestione degli utenti permette di particolareggiare l'accesso a determinate risorse tramite la configurazione dei privilegi di accesso (Permessi).
Ogni file e processo ha un insieme di flag che permette di gestire tale politica.



# Locale e Remoto

Definizioni:

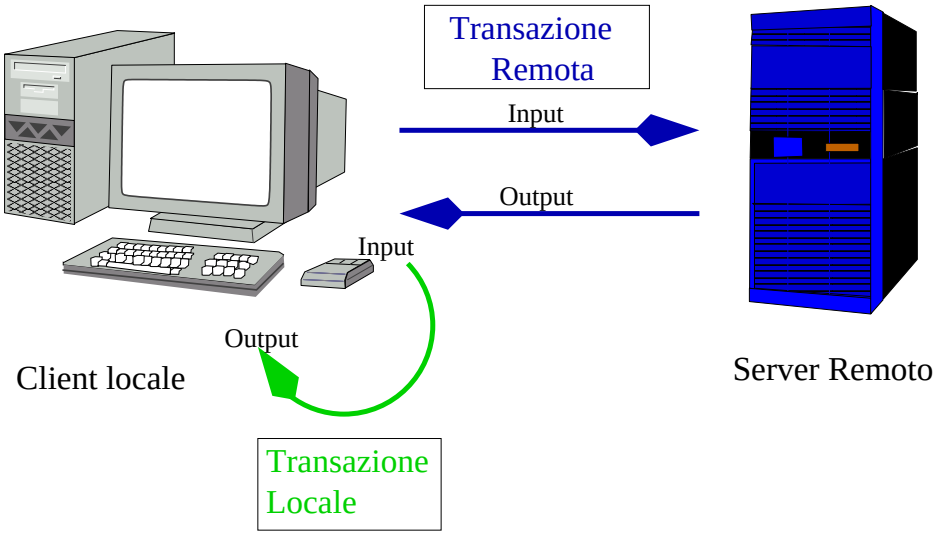
- **Locale**: L'input viene elaborato direttamente sulla macchina fisicamente connessa alla mia tastiera e monitor che restutuisce un output.
- **Remoto**: L'input viene elaborato dalla macchina alla quale ho accesso.



# Client e Server

Definizioni:

- **Client**: La macchina che richiede un servizio *per mezzo di un protocollo*. L'input corrisponde alla richiesta.
- **Server**: La macchina che elargisce un servizio *per mezzo di un protocollo*. L'output corrisponde al risultato di tale servizio.



# HTTP Cenni

HTTP - HyperText Tranfer Protocol

- Si comporta come un sistema Input-Output regolato da determinate specifiche dette *Specifiche di Protocollo* definite in *www.w3.org/Protocols/rfc2616/rfc2616.html*.
- Protocollo Client-Server.
- È un protocollo di rete usato per inviare praticamente ogni tipo di file e altre risorse.
- Un *browser* è un client HTTP che invia una richiesta (Request) ad un server HTTP.
- Un *server HTTP (server web)* restituisce una risposta (Response) al browser.

# HTTP Cenni

- **Richiesta del client.**

1. GET / HTTP/1.1
2. User−Agent: Mozilla/4.0
   Host: 127.0.0.1:80
   Accept: text/html, */*
   Accept−Charset: iso−8859−1;q=1.0
   Accept−Encoding: deflate, gzip, x−gzip, identity, *;q=0
   Connection: Keep−Alive

1. Metodo di richiesta della risorsa. 2. Header

- **Responso del server.**

1. HTTP/1.1 200 OK
2. Date: Fri, 07 Nov 2003 08:53:17 GMT
   Server: Apache/1.3.28 (Unix) PHP/4.3.3
   Content−Location: index.html.en
   Vary: negotiate,accept−language,accept−charset
   TCN: choice
   Last−Modified: Fri, 04 May 2001 00:00:38 GMT
   ETag: "66d2c−5b0−3af1f126;3fa42c03"
   Accept−Ranges: bytes
   Content−Length: 1456
   Content−Type: text/html
   Content−Language: en
3. **Corpo del Responso**

1. Status Line. 2. Header. 3. Corpo del responso

# Ricordati di me: Cookies

Il protocollo HTTP è un protocollo senza memoria (stateless), il che signifi ca che non ci fornisce nessun metodo integrato per mantenere gli stati di una connessione durante la navigazione di ogni singolo utente.
Ogni singola connessione fa storia a sè.
Per creare una memoria dell'utente che si connette in momenti diversi ad un sito o per creare una continuità durante la navigazione su più pagine dello stesso sito sono stati inventati i *cookies* fornite dagli stessi server web o dalle applicazioni web. In realtà i cookies sono dei fi le che vengono memorizzati in memoria o su disco e contengono dati eterogenei che il server, una volta richiesto il cookie elaborerà a suo piacimento.

# Ricordati di me: Sessioni

L'idea che sta alla base di tali tecniche è che il server genera un *identifi catore di sessione* all'inizio della connessione con l'utente e lo invia al browser il quale ne restituirà il valore ogniqualvolta il server ne farà richiesta.
Da questo punto in poi il server è sicuro che tale identifi catore sarà lo stesso durante un periodo predefi nito della navigazione tra le pagine da parte dell'utente.
Le sessioni sono quindi delle stringhe appositamente generate per identifi care gli utenti e possono essere usati dal server per mantenere e rendere accessibili i dati inerenti ogni sessione (eg. variabili).

Lo stesso discorso vale per i cookies.

# Linguaggi Lato Client

- *Html (HypeText Markup Language):* è un linguaggio che viene generato e inviato dal server web e viene interpretato dal browser per generare un layout di pagina inserendo immagini testo ed altro. È un linguaggio esclusivamente statico, non prevede cioè l'interazione con l'utente.
- *Javascript:* è un linguaggio che viene di norma inserito nelle pagine html mandate dal server web ed eseguito dal browser. Permette di creare un certo livello di dinamicità alle pagine Html permettendo di farvi interagire l'utente.

# Vulnerabilità e Accesso alle Risorse 1/2

Consideriamo i punti di vista:

- Vulnerabilità locali - Punto di vista del client
  - **Cross Site Scripting et al.:** Permette, attraverso l'inserimento di codice javascript all'interno di una pagina Html inviata dal server, di fare eseguire dal client delle azioni non previste. *È una vulnerabilità lato server che ha effetto sul client.*
  - **Esecuzione arbitraria di comandi:** Permette di eseguire codice che può portare alla compromissione della macchina locale. *È una vulnerabilità del client.*
  - **Accesso ai dati:** Permette di avere accesso a file sulla macchina locale. *È una vulnerabilità del client.*

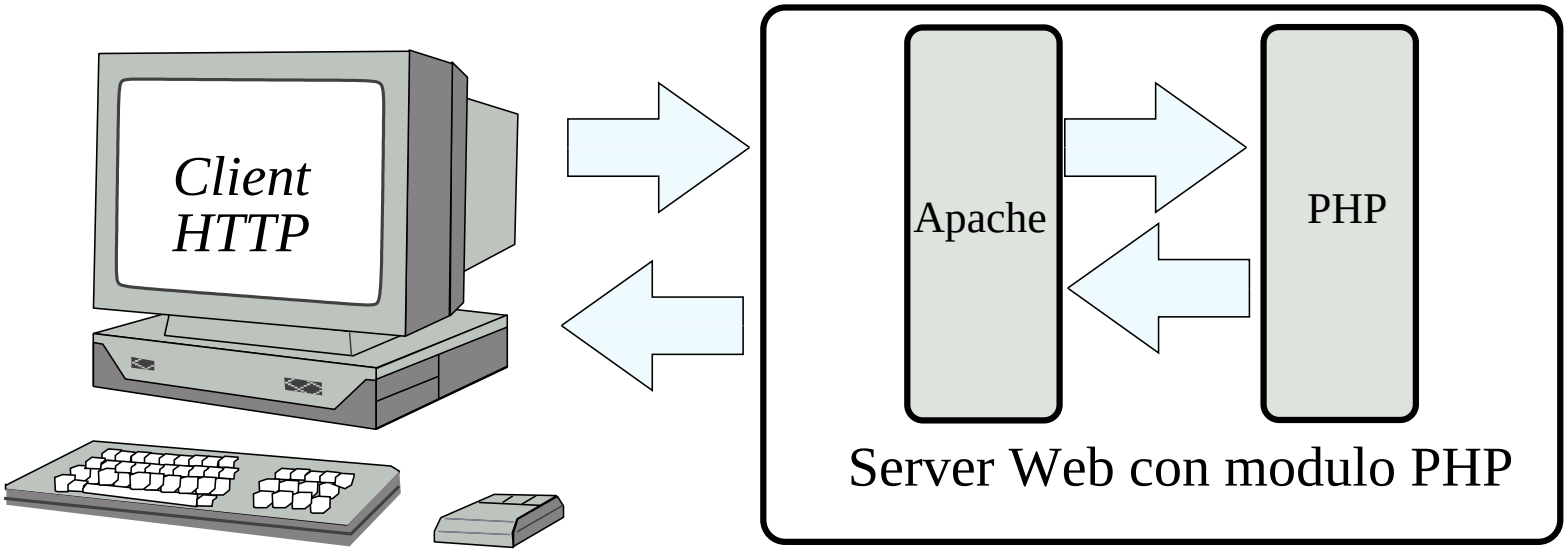
# Vulnerabilità e Accesso alle Risorse 2/2

Consideriamo i punti di vista:

- Vulnerabilità remote - Punto di vista del server
  - **Accesso ai dati:** Permette di avere accesso a file sulla macchina remota. *È una vulnerabilità lato server.*
  - **Esecuzione arbitraria di comandi:** Permette di eseguire codice sulla macchina remota. *È una vulnerabilità lato server.*
  - **Negazione dei servizi (DoS):** Blocca uno o più servizi, impedendo al server di avere una normale attività. *È una vulnerabilità lato server.*

# *PHP*

# PHP - Linguaggi Lato Server e Apache



# PHP - Per Cominciare: php.ini

Php.ini è il fi le di confi gurazione del processore PHP.

La sintassi di assegnazione di valori a una variabile è:

Variabile = Valore

**I settaggi delle variabili per la sicurezza:**

| Variabile | Sicuro | Insicuro |
|---|---|---|
| register_globals | On | Off |
| magic_quotes_gpc | On | Off |
| expose_php | Off | On |
| allow_url_fopen | Off | On |
| file_uploads | Off | On |
| upload_tmp_dir | * | ** |
| log_errors | On | Off |
| display_errors | Off | On |
| error_log | * | ** |
| safe_mode | On | Off |
| include_path | * | ** |

*** Percorso fuori dalla directory radice del server web**

**** Percorso dentro la directory radice del server web**

# PHP - I settaggi delle variabili per la sicurezza:

**register_globals=On**

In PHP le variabili inviate tramite una richiesta del client possono essere accessibili direttamente dal codice php.

Questa direttiva dice all'interprete PHP se considerare globali le variabili assegnate nella richiesta del client.

Per esempio:

```
Se register_globals = On
la richiesta "GET http://www.example.com/test.php?id=3"
e test.php:
<? print "<html> id e' uguale a ".$id."</html>" ?>
produrra' un output del tipo:
<html> id e' uguale a 3 </html>
Se register_globals = Off
test.php produrr\'a un output del tipo:
<html> id e' uguale a  </html>
Per produrre in output il valore di id, test.php dovra' essere scritto:
<? print "<html> id e' uguale a ".$_GET['id']."</html>" ?>
```

# PHP - I settaggi delle variabili per la sicurezza:

**magic_quotes_gpc=On**

Ad ogni eventuale occorrenza di ’ " \ e il carattere nullo PHP ci aggiunge automaticamente uno \ .

```
Se  magic_quotes_gpc= On
la richiesta "GET http://www.example.com/test.php?id=’ciao’"
e test.php:
<? print "<html> id e’ uguale a ".$id."</html>" ?>
produrra’ un output del tipo:
<html> id e’ uguale a \’ciao\’ </html>
Se magic_quotes_gpc = Off
test.php produrra’ un output del tipo:
<html> id e’ uguale a ’ciao’ </html>
```

# PHP - I settaggi delle variabili per la sicurezza:

**expose_php=Off**

Questa direttiva dice al processore PHP di non stampare all'interno degli header della risposta del server web la presenza del PHP.

```
Se  expose_php = On
la richiesta "GET http://www.example.com/test.php"
.....
Server: Apache/1.3.28 (Unix) PHP/4.3.3
.....
Se  expose_php = Off
.....
Server: Apache/1.3.28 (Unix)
.....
```

# PHP - I settaggi delle variabili per la sicurezza:

**allow_url_fopen=Off**

La funzione *fopen* permette di avere accesso ad un fi le o una risorsa. Può aprire anche fi le remoti se il fi le da aprire è del tipo "http://url" o "ftp://url".

**file_uploads=On**

Tale direttiva permette al motore PHP di gestire direttamente gli upload dei fi le.

**log_errors=On**

Tale direttiva permette al motore PHP di scrivere in un fi le gli errori generati da eventuali attacchi oltre che da eventuali errori nel codice.

**display_errors=Off**

Mostra gli errori generati dal PHP direttamente nella pagina di risposta del server web.

Questo potrebbe permettere eventuali utenti maliziosi di ottenere informazioni utili.

# PHP - I settaggi delle variabili per la sicurezza:

**safe_mode=On**

| Variabile | Sicuro | Insicuro |
|---|---|---|
| safe_mode | On | Off |
| safe_mode_gid | On | Off |
| safe_mode_include_dir | * | ** |
| safe_mode_exec_dir | * | ** |
| safe_mode_allowed_env_vars | ” | |
| safe_mode_protected_env_vars | ” | |
| open_basedir | ” | |
| disable_functions | ” | |
| disable_classes | ” | |

*** Percorso fuori dalla directory radice del server web**
**** Percorso dentro la directory radice del server web**

# PHP - Filtraggio degli Input

Per un protocollo come l'HTTP gli input che il server riceve sono all'interno della *Client Request*, quindi l'input più importante da filtrare sarà **tutto** quello che arriva dal client:

- La risorsa: http://www.example.com/test.php
- I Cookies.
- Le variabili nell'intestazione HTTP (HTTP Header): User-Agent, Referer...

Ma cosa bisogna filtrare?

# PHP - Filtraggio degli Input

**Esempio:**

Supponiamo di voler includere una pagina php nel nostro codice test.php. Allora usiamo:

```
<?
include($page);
print "prova\n";
?>
```

dove $page sarà una variabile accessibile da un link nella nostra pagina html.
Allora se inviamo la seguente richiesta lato client:

```
GET http://www.example.com/test.php?page=/etc/passwd HTTP/1.0
```

Otterremo l'inclusione del fi le passwd che risiede fuori dalla directory radice del server web.

Tale fi le verrà così incluso da test.php nella pagina html generata e inviata al client.

# PHP - Filtraggio degli Input

**Soluzione dell'esempio:**

Consideriamo il seguente codice.

```
<?
$page=ereg_replace("\\","",$page);
include($_SERVER['DOCUMENT_ROOT']."/include/my_".$page.".php");
print "prova\n";
?>
```

Se inviamo la seguente richiesta lato client:

```
GET http://www.example.com/test.php?page=/etc/passwd HTTP/1.0
```

Non otterremo più l'inclusione del fi le passwd ma solo una pagina vuota.
Infatti il codice in test.php cercherà di aprire il seguente fi le

`/var/www/htdocs/include/my_etcpasswd.php}.`

Abbiamo fi ltrato l'input.

# PHP - Filtraggio degli Input

Per filtrare al meglio i dati in ingresso bisogna fare una distinzione tra **i caratteri e i meta-caratteri:**

- I caratteri sono tutti quei simboli che non hanno nessun altro significato (Es. a-z,A-Z,0-9).
- I meta-caratteri sono tutti quei simboli (stampabili e non) il cui significato dipende dal contesto (Es. lo / è anche il separatore che individua il percorso di un file).

Nel momento in cui stabiliamo in quale contesto lavoreremo e cosa effettivamente si vuole che il sistema faccia, tutti gli altri caratteri dovranno essere filtrati.

# PHP - Le potenziali Vulnerabilità

**SQL Injection:**

Supponiamo di avere il seguente codice html:

```
<form action="login.php" method="get">
User name:
<input type="text" name="Username"/>
Password:
<input type="password" name="Password"/>
<input type="submit" value="OK" />
</form>
```

Possiamo assumere che $Username e $Password siano usati in una query SQL come questa:

```
SELECT * FROM UserTable WHERE Username='$Username'
AND Password='$Password'
```

# PHP - Le potenziali Vulnerabilità

**SQL Injection:**

Il nostro codice php

```
<?
  $socket=mysql_connect("localhost","root","password")
   or die("errore ".mysql_error());
  mysql_select_db("User_Login");
  $query="SELECT * FROM UserTable WHERE Username='$Username'
   AND Password='$Password";
  $result=mysql_query($query) or die("errore ".mysql_error());
  $num_fields = mysql_num_fields($result);
  if($num_fields)
  {
   print "Welcome ".$Username;
  }
?>
```

# PHP - Le potenziali Vulnerabilità

**SQL Injection:**

Se il client invia la seguente richiesta:

```
GET http://www.example.com/login.php?Username=admin'%3b--&Password=gfdfs
```

login.php interpreterà la variabile $query nel seguente modo:

```
SELECT * FROM UserTable WHERE Username='admin'; --' AND Password='fdgd'"
```

e la invierà al server mysql che, non trovando errori la eseguirà ritornando un valore di $num_fi elds maggiore di zero!!

Siamo entrati come admin!

# PHP - Le potenziali Vulnerabilità

**SQL Injection:** In realtà quello che si potrebbe fare un una query sql ce lo può dire solo il DBMS al quale chiediamo di eseguirla.
MySql permette di:

- Caricare il contenuto di un fi le in una tabella:

  ```
  LOAD DATA INFILE 'file' INTO TABLE 'tabella';
  ```

- Inserire il contenuto di un fi le in un campo di una tabella:

  ```
  INSERT INTO 'tabella' 'campo' VALUES (LOAD_FILE('file'));
  INSERT INTO 'tabella' ('campo') VALUES (LOAD_FILE(CHAR(##,..)));
  ```

- Scrivere in un fi le il contenuto di una tabella:

  ```
  SELECT * from 'tabella' INTO OUTFILE 'file';
  ```

Tutti metodi che possono portare alla compromissione totale del server!!

# PHP - Le potenziali Vulnerabilità

**SQL Injection:** Come difendersi?

1. Verifi care che le variabili in php.ini abbiano i valori giusti per la sicurezza. In questo modo saremmo già a posto (ci pensa magic_quotes_gpc)
2. Pochè non si sa mai è sempre meglio fi ltrare ogni singolo carattere delle variabili che riceveremo dalla form Html.

# PHP - Le potenziali Vulnerabilità

**Cross Site Scripting (XSS):**

Consideriamo sempre il nostro codice php

```
<?
  $socket=mysql_connect("localhost","root","password")
                          or die("errore ".mysql_error());
  mysql_select_db("User_Login");
  $query="SELECT * FROM UserTable WHERE Username='$Username'
                                       AND Password='$Password";
  $result=mysql_query($query) or die("errore ".mysql_error());
  $num_fields = mysql_num_fields($result);
  $expire=time()+60*60*24*30;
  get_cookie("example.com",$Username.$Password,$expire);
  if($num_fields) {
   print "Welcome ".$Username;
  set_cookie("example.com",$Username.$Password,$expire);
  }else{
  print "Spiacenti. Nessun utente con nome ".$Username;
  }
?>
```

# PHP - Le potenziali Vulnerabilità

**Cross Site Scripting (XSS):**

Se il client invia la seguente richiesta:

```
GET http://www.example.com/login.php?Username=<script>
          alert(document.cookie);</script>&Password=null
```

login.php stamperà il valore di Username:

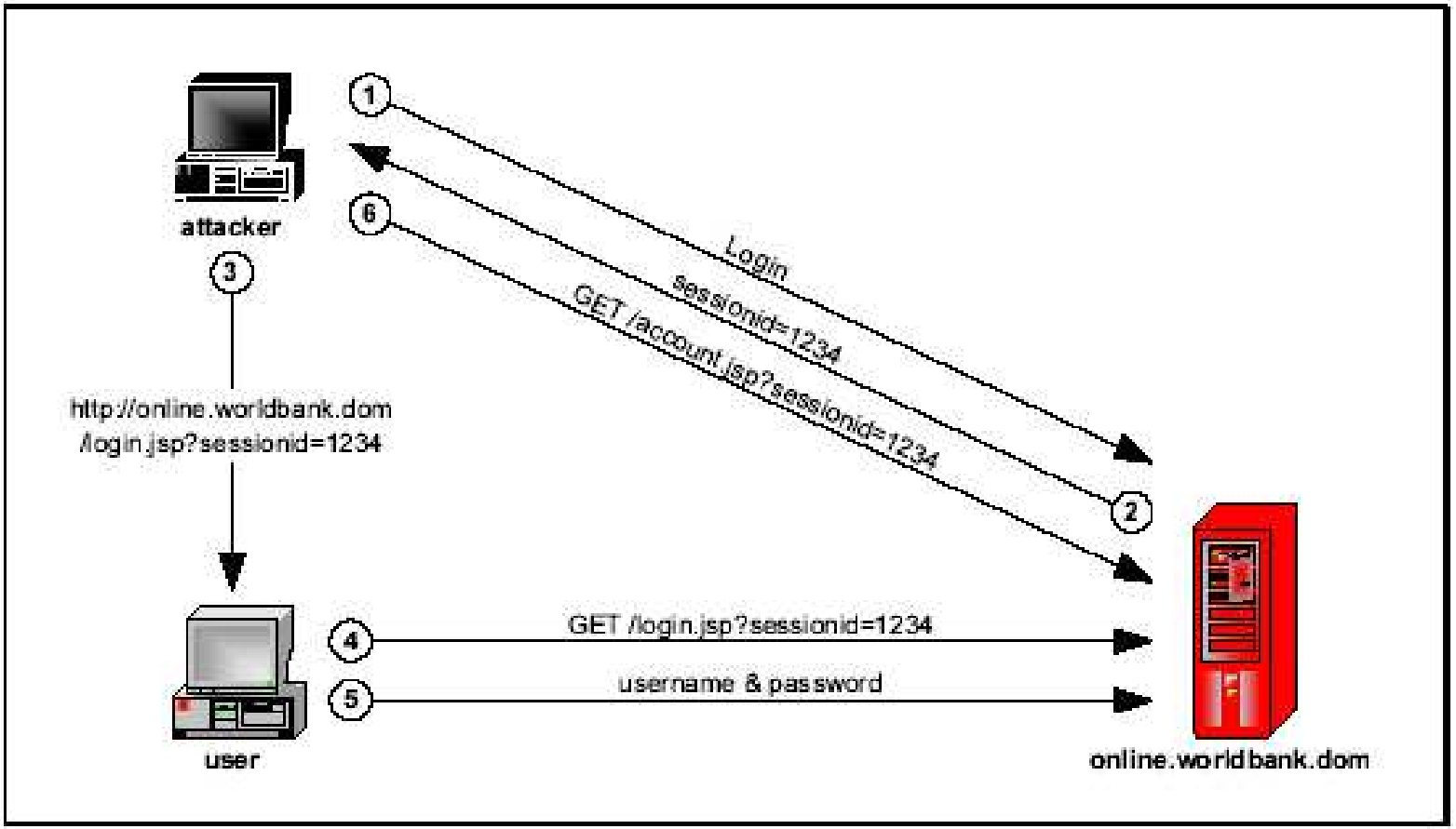
```
Spiacenti. Nessun utente con nome
        <script>alert(document.cookie);</script>
```

Che verrà interpretato dal browser ed eseguirà lo script, visualizzando una fi nestra con

stampato l’eventuale cookie contenente eventuale Username e password.

# PHP - Le potenziali Vulnerabilità

## XSS e Session Hijacking/Fixation:



# PHP - Le Funzioni ’Pericolose’

In tutti i linguaggi di programmazione le funzioni da definire ’pericolose’, sono tutte le funzioni che ad un input generano un output!!
In PHP le funzioni più ’pericolose’ sono:

- **Inclusione di file:** include(), require(), include_once(), eval();
- **Apertura di file:** *open();
- **Scrittura di file:** *puts();
- **Esecuzione di processi:** exec(), “ (apici inversi), system(), passthru(), proc_open(), shell_exec();
- ...e molte altre!!

Ovviamente, tutto dipende da che *sottoinsieme di input* si da a tali funzioni....

# PHP - Le Funzioni ’Amiche’

In PHP vi sono delle funzioni che ci vengono in aiuto per fi ltrare gli input.

- **string addcslashes ( string str, string charlist )**
- **string quotemeta ( string str )**
- **string htmlspecialchars ( string string [ , int quote_style [ , string charset ] ] )**
- **string htmlentities ( string string [ , int quote_style [, string charset ] ] )**
- **string preg_quote ( string str [ , string delimiter ] )**
- **mixed preg_replace ( mixed pattern, mixed replacement, mixed subject [, int limit ] )**

# *Mysql*

# MySql - Opzioni di Inizializzazione

Nel file di configurazione **my.cnf** si ha la possibilità di intervenire sulla sicurezza del DB.

- **skip-networking**: se è attivo non fa uso del TCP/IP.
- **user**: Il DBMS gira come utente a bassi privilegi (di solito mysql).

# MySql - Utenti e Privilegi

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY [PASSWORD] 'password']
      [, user_name [IDENTIFIED BY [PASSWORD] 'password'] ...]
  [REQUIRE
       NONE |
       [{SSL| X509}]
       [CIPHER cipher [AND]]
       [ISSUER issuer [AND]]
       [SUBJECT subject]]
  [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
       MAX_UPDATES_PER_HOUR # |
       MAX_CONNECTIONS_PER_HOUR #]]
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    FROM user_name [, user_name ...]
```

grant select(colonna) on tabella to user@localhost identifi ed by 'password' ;

# MySql - Le Funzioni Pericolose

- LOAD DATA (LOCAL)
- SELECT INTO OUTFILE
- CHAR()
- LOAD FILE()

# MySql - I dati

**Le Funzioni di crittazione dei dati:**

- MD5(),
- SHA1(),
- PASSWORD().

# MySql - Linee Guida per un DB ’Sicuro’

- Non usare utenti anonimi.
- Creare un DB accessibile solo ad un numero ristretto i utenti
- Dare ad ogni utente che ha accesso al DB solo ed esclusivamente i privilegi necessari.
- Considerare l’eventualità di fare girare il DB solo in locale.
- Utilizzare sempre password criptate.

# Riferimenti

- http://www.google.it/search?q="php+security"
- http://www.php.net/manual/en/features.safe-mode.php
- http://www.mysql.com/doc/en/Security.html
- http://www.mysql.com/doc/en/Privileges.html
- http://www.acros.si/papers/session_fixation.pdf

*Grazie! E ricordate non fidatevi di quello che vi si dice!*