securenetwork

il percorso più sicuro.

Sicurezza delle applicazioni web

(Programmazione sicura in ambiente web)

Luca Carettoni – l.carettoni@securenetwork.it

26 Novembre – Linux Day 2005

©2005 – Secure Network S.r.l.

Il peggiore dei mondi...

- Un server web è implicitamente pensato per rispondere a richieste che provengono dall'esterno

- I servizi sono offerti ad utenti generici, ricevendo e gestendo degli input su un canale tendenzialmente inaffidabile

- Il protocollo è stateless: non c'è uno stato persistente

- Tutte le aziende vogliono essere in rete

©2005 – Secure Network S.r.l.

Sicurezza nelle web application

Un requisito importante, al pari di quelli funzionali, che spesso viene trascurato...

• Applicazioni web NON siti web !

• Le applicazioni diventano sempre più aperte e flessibili: garantire il paradigma CIA non è un compito semplice

• L'aspetto sicurezza è il primo a “saltare” quando la deadline si avvicina

Risultato: @stake stima che il 70% delle applicazioni analizzate presenta vulnerabilità rilevanti.

©2005 – Secure Network S.r.l.

Sicurezza nelle web application

La sicurezza dipende da molteplici fattori:

- Software bacato e/o malconfigurato (es: “unicode bug”)
- Ambiente insicuro (es: packet sniffing)
- Logica applicativa progettata/sviluppata male

Noi ci occuperemo di quest'ultimo aspetto, presentando DIECI REGOLE da ricordare (e mostrando cosa si rischia non rispettandole!)

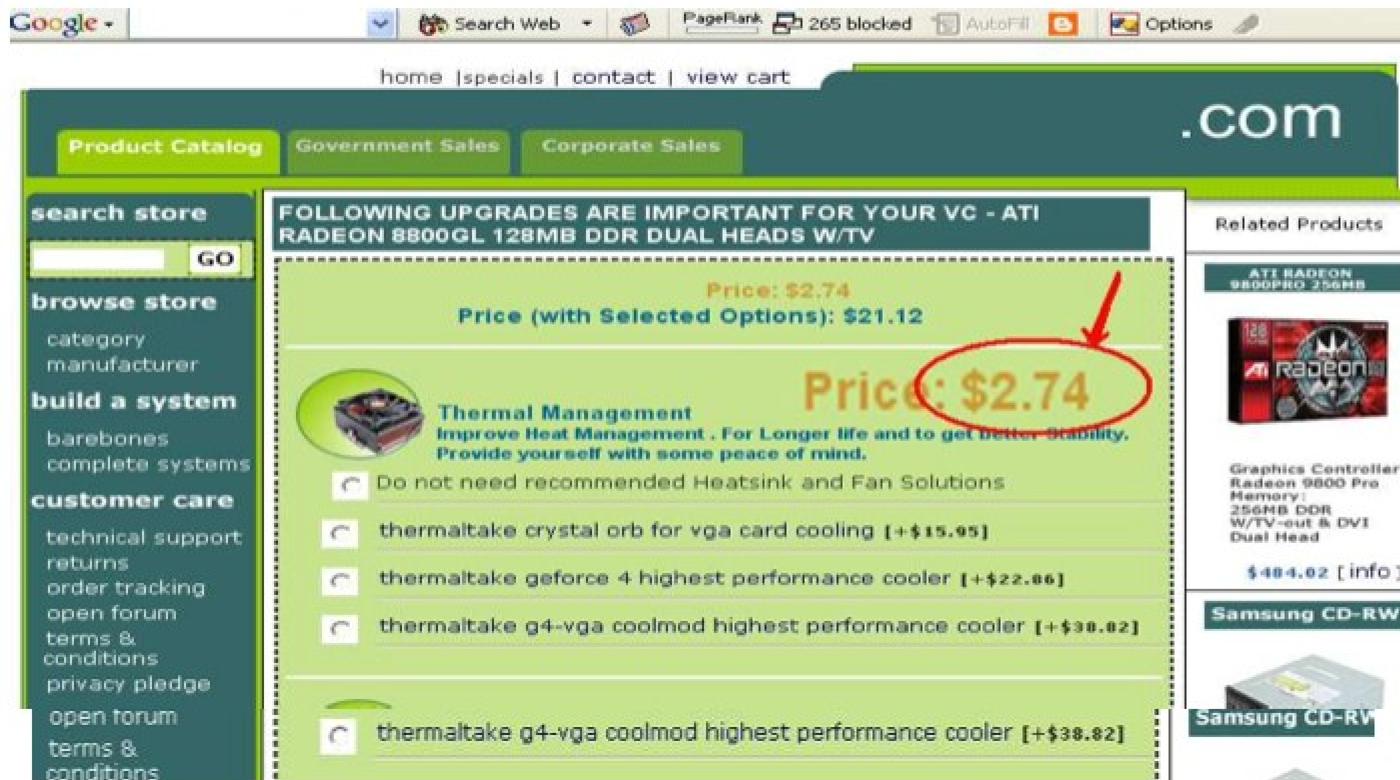
©2005 – Secure Network S.r.l.

1) Non esiste sicurezza sul lato client

Campi html HIDDEN

©2005 – Secure Network S.r.l.

1) Non esiste sicurezza sul lato client



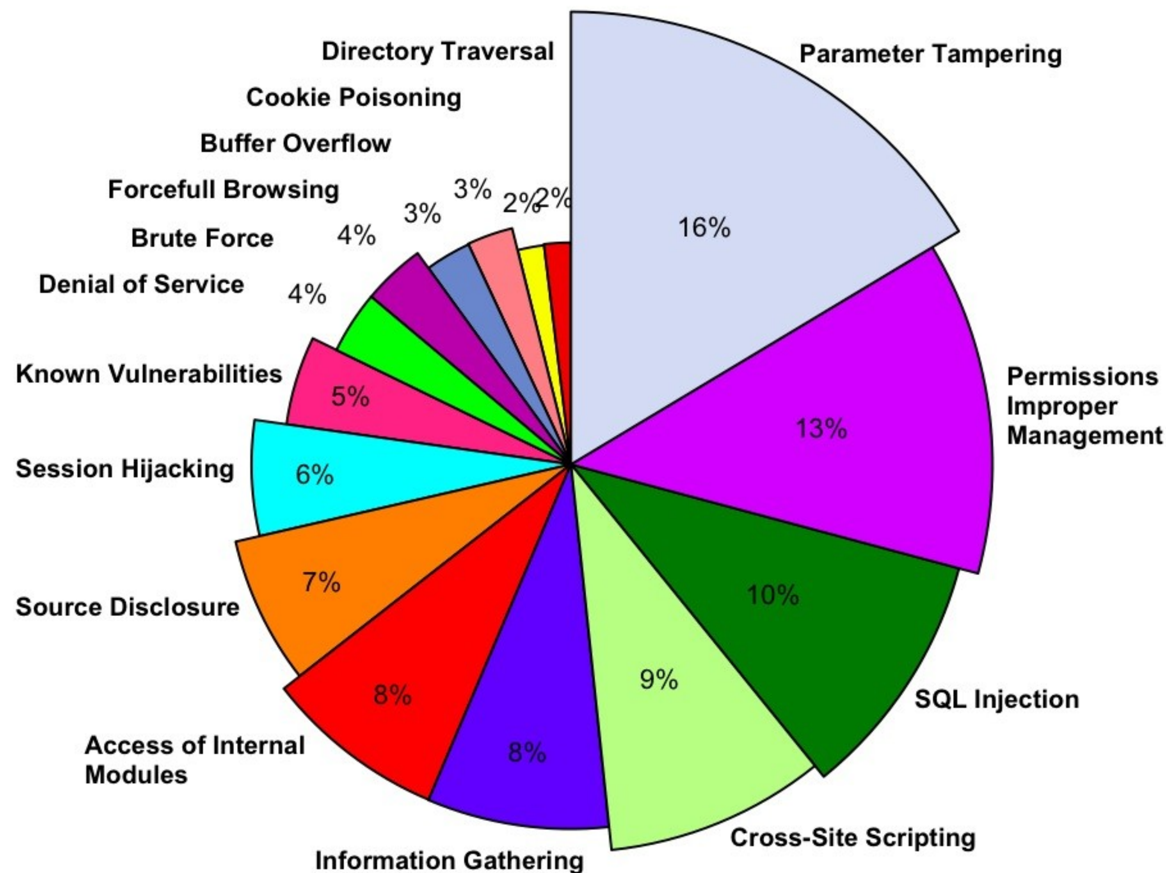
I dati lato client devono essere considerati inaffidabili.

Mai fidarsi di:
- validazione dell'input tramite javascript
- variabili REFERER e simili

©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

La causa principale dei problemi di sicurezza nelle applicazioni web deriva dalla mal validazione dell'input



©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Un esempio reale (Squirrel Mail):

```
$day=$_GET['day'];
$month=$_GET['month'];
$year=$_GET['year'];
echo"<a href=\"day.php?year=$year&amp;"
echo"month=$month&amp;day=$day\">";
```

Se la richiesta è:

```
event_delete.php?year=><script>myCode();</script>
```

L'html di risposta contiene:

```
<html>
<body>
....
<a href="day.php?year=><script>myCode();</script>
```


©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Un perfetto attacco Cross-Site Scripting (XSS) !!!
Ottimo preludio per il peggio: ***session hijacking***.

Come spesso accade la soluzione è più che banale:
is numeric($ GET['month'])

Questa è la modifica presente nelle versioni attuali del software

Ma cosa dobbiamo considerare?
Tanti caratteri tendenzialmente pericolosi, tanti standard, tanti livelli di interpretazione a cui sono associati metalinguaggi diversi

©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Nella nostra applicazione non vogliamo permettere l'inserimento di tag html/javascript (come il caso illustrato)

Q: Perchè non rimuovere solamente l'uso del tag **<SCRIPT>**?

A: Perchè esistono altri cento modi per passare uno script javascript (*Doh!*)

<IMG SRC="javascript:alert('Ciao');">
<ANYTHING SRC="javascript:alert('Ciao');">

filtro la parola chiave “javascript” ma anche così funziona:

<IMG SRC="javasc

ript:alert('Ciao');">

©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Allora filtro tutto (CR-LF, CR, tab, spazi)
Peccato che non abbiamo considerato i vari tipi di codifica:

**<IMG SRC="javasc	ript:alert ('Ciao');">**

**<IMG SRC="javasc
ript:alert ('Ciao');">**

**<IMG SRC=javasc
ript:alert ('Ciao');>**

Ovviamente poi tutto va pensato per i caratteri maiuscoli.

©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Un altro esempio reale (Squirrel Mail+Addressbook plugin):

Struttura della tabella che andremo a considerare:

```
CREATE TABLE address (
owner varchar(50) default NULL,
nickname varchar(50) default NULL,
firstname varchar(50) default NULL,
lastname varchar(50) default NULL,
email varchar(50) default NULL,
label varchar(50) default NULL )
```

All'interno della pagina
squirrelmail/squirrelmail/functions/abook database.php
troviamo:

```
$query = sprintf("SELECT * FROM %s WHERE owner=’%s’ AND
nickname=’%s’", $this->table, $this->owner, $alias);
$res = $this->dbh->query($query);
```


©2005 – Secure Network S.r.l.

2) Validare tutto l'input, sempre!

Inutile dire che in nessun altro punto del codice viene validato il campo denominato *alias*

Una richiesta del tipo:
’ UNION ALL SELECT * FROM address WHERE ’’=’

trasforma la query nella seguente:

SELECT * FROM address WHERE owner=’me’ AND nickname=’’
UNION ALL SELECT * FROM address WHERE ’’=’’

- Union tra due query SQL compatibili
- Stesso numero di parametri

Ecco un esempio di ***Sql Injection*** !!!

©2005 – Secure Network S.r.l.

3) Non fornire messaggi di errore dettagliati

Per poter portare a termine un'aggressione informatica servono molte informazioni sul sistema:
- Servizi presenti
- Versioni
- Configurazioni
- Path

Rivelare in maniera diretta parte di queste informazioni è un errore apparentemente innocuo ma che spesso fa la differenza

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]
All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists

©2005 – Secure Network S.r.l.

3) Non fornire messaggi di errore dettagliati

Ricordate: passate sempre errori generici una volta eseguito il deployment su web della vostra applicazione

Inoltre è bene:

- Evitare di mostrare messaggi di errore che forniscano più informazioni del dovuto

“Username/Password sono errati”

vs

“Errore nella password per username xxxx”

©2005 – Secure Network S.r.l.

4) Generare sempre nuovi ID di sessione

• Session ID gestiti lato server
Fidarsi ciecamente del valore dei cookie è male

• Session ID diversi tra HTTP e HTTPS all'interno della stessa applicazione

• Session ID nuovi nel caso di aumento di privilegi

• Session ID sicuri (non “guessabili”)

©2005 – Secure Network S.r.l.

4) Generare sempre nuovi ID di sessione

Come funziona l'attacco denominato “Session Fixation”:

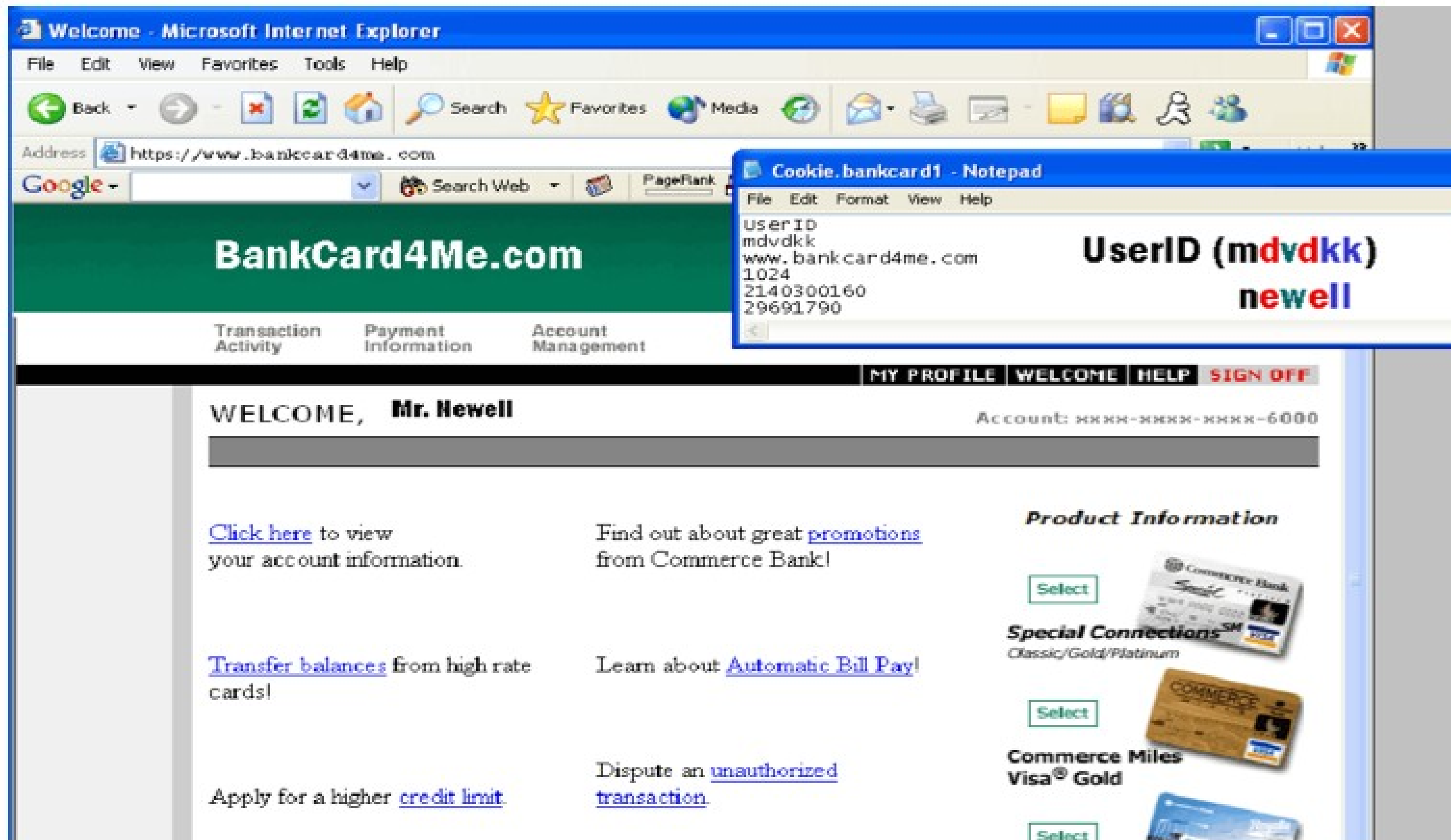
1. L'aggressore ottiene un Session ID valido (tramite una normale richiesta)

2. In qualche modo (es:XSS), fornisce alla vittima una trap session
login.php?PHPSESSID=*SessionIDAttacker*

3. Se il sito supporta la definizione di Session ID nell'url, la vittima userà tale identificatore durante la sua sessione.
Il token però è condiviso con l'aggressore

©2005 – Secure Network S.r.l.

4) Generare sempre nuovi ID di sessione (...sicuri possibilmente!)



©2005 – Secure Network S.r.l.

5) Ridondanza, sempre!

Spesso qualcosa non funziona, spesso ci si dimentica di alcuni dettagli, spesso la legge di Murphy ha il predominio sulla situazione...

Security “in depth”:
- permessi sul filesystem
- privilegi sul database
- rivalidazione dell'input
- configurazione opportuna dell'infrastruttura

©2005 – Secure Network S.r.l.

6) Usare sempre WHITELIST

L'uso di **WHITELIST** per la validazione dell'input si basa sulla definizione di valori “ammissibili” per la nostra applicazione.

L'uso di **BLACKLIST**, al contrario, è invece basato sulla definizione di valori non conformi alle specifiche;
Per valori “non ammissibili” intendiamo input che possono generare errori anche a livello funzionale oltre che strettamente legati alla sicurezza.

Spesso (purtroppo) non sappiamo definire in maniera precisa cosa scartare.
Inoltre le applicazioni vengono continuamente modificate e reintegrate.

©2005 – Secure Network S.r.l.

7) Non “plasmare” l'input dell'utente

Gli utenti sono spaesati davanti ad un errore applicativo;
Gli sviluppatori cercano di evitare tali situazioni.

Capendo la logica del programma, capendo la logica dell'handling si può imbrogliare (ancora una volta) il software:

http://www.example.com/getInfo.asp?file=paolo.txt

Noi però vogliamo evitare problemi di *directory traversal*
http://www.example.com/getInfo.asp?file=../../boot.ini
aggiungendo il seguente codice:

filename = Request.QueryString(“file”);
Replace(filename, “/”,”\”);
Replace(filename, “..\”,””);

©2005 – Secure Network S.r.l.

7) Non “plasmare” l'input dell'utente

L'utente “furbo” potrebbe però inoltrare la seguente richiesta:

http://www.example.com/getInfo.asp?file=....//....//boot.ini

che viene così trasformata

....\\....\\boot.ini

..\..\boot.ini

Proprio quello che ci serviva !

Nota: Windows accetta come path separator sia “\” che “/”

©2005 – Secure Network S.r.l.

8) Non salvare mai dati critici in chiaro

Durante la fase di login il confronto tra la password immessa dall'utente e quella precedentemente salvata sul sistema deve essere fatta tramite digest.

La funzione di hashing MD5 è una buona soluzione:
- One way hash function deterministica
- Digest di dimensione fissa (128bit)
- Presente come funzione di libreria in tutti i linguaggi

$passHash=md5($pass)

“Ciao” diventa **16272a5dd83c63010e9f67977940e871**

©2005 – Secure Network S.r.l.

8) Non salvare mai dati critici in chiaro

In questo modo anche se il database che contiene gli account viene compromesso (per esempio tramite *Sql Injection*), l'aggressore non può recuperare le credenziali di login.

Perciò DIFFIDATE dai portali web che forniscono la funzione “Password Dimenticata” inviandovi per email la password scelta durante la registrazione.

Stesse considerazioni valgono per la memorizzazione di dati critici...cifrare tutto ciò che è importante!

Per inciso: una password sul file *.MDB non significa cifrare !!!

©2005 – Secure Network S.r.l.

9) C'è sempre qualcuno più “furbo”

- Non possiamo prevedere la capacità tecnica dell'aggressore. Quello che reputiamo “sicuro” potrebbe non esserlo

- Spesso c'è una sola implementazione sicura e altre cento funzionalmente equivalenti ma che presentano qualche problema. Riconoscere la soluzione “giusta” non è banale

- L'aggressore ha molta più libertà d'azione rispetto a quanto pensiamo (social engineering, compromissione di sistemi partner, attacchi dalla intranet)

- Pair programming e auditing svolto da personale esperto può ridurre il numero delle vulnerabilità nel software

©2005 – Secure Network S.r.l.

10) La sicurezza è un processo, non un prodotto [Bruce Schneier]

• Non pensare mai di essere “arrivati”

• Essere consapevoli che non possiamo annullare completamente il rischio di un'aggressione

La sicurezza è una caratteristica importante del software. Non pensare ad essa solo al termine dello sviluppo

Come fare, allora?

- Valutate cosa dovete proteggere
- Valutate tecnicamente/economicamente come proteggere
- Implementate il sistema di sicurezza
- Formazione, Verifica e Controllo periodico

©2005 – Secure Network S.r.l.

Concludendo

"Non esiste una pallottola d'argento per la sicurezza in ambito web, anche se alcune società del settore tendono a farlo credere.

Per risolvere questa sfida occorrono persone capaci, grande conoscenza e ottima formazione, ottimi processi e il meglio della tecnologia. "

Mark Curphey - OWASP founder

©2005 – Secure Network S.r.l.

Grazie per l'attenzione

Luca Carettoni - l.carettoni@securenetwork.it

Homepage: **www.ikkisoft.com**
SecureNetwork S.r.l.: **www.securenetwork.it**

©2005 – Secure Network S.r.l.