



# *Sicurezza delle applicazioni web*

Ing. Stefano Zanero  
17/05/2006

# La natura di HTTP è un problema

- Stateless protocol: non c'è uno stato persistente (problemi per sessioni)
- Un server web è implicitamente pensato per rispondere a richieste che arrivano dall'esterno... spesso, da chiunque
- I web server spesso presentano delle loro vulnerabilità: su questo argomento c'è una certa sensibilità (anche se non abbastanza)

# Le applicazioni web sono un problema

- Secondo @stake, il 70% delle applicazioni revisionate da loro presenta difetti di sicurezza rilevanti
- La sicurezza è spesso l'ultimo dei risvolti considerati durante lo sviluppo
- Il re-design delle applicazioni web è un problema di tempo e di costo
- L'interazione tra i comportamenti del server, eventuali framework, codice di terze parti e codice “custom” crea vulnerabilità

# Un ambiente pericoloso

- Internet è un ambiente pericoloso (e lo sappiamo)
- Cosa fa una web application? Offre servizi ad utenti generici, ricevendo e gestendo degli input su un canale tendenzialmente inaffidabile
- Siamo esattamente nel *peggiore dei casi possibili, e questo è il modello computazionale verso cui ci stiamo spostando... auguri !*

# Il client non è affidabile!

- E quindi non *dovremmo fidarci...*
  - *Delle validazioni di campi fatte tramite javascript*
  - *Delle variabili REFERER e simili passate dal client*
  - *Del fatto che le form siano state sottomesse proprio dalla pagina che noi abbiamo disegnato, e non da una “variante”...*

# Un esempio banalissimo...

product1449[1] - Notepad

```
File Edit Format View Help
</tr>
<tr>
  <td valign="top"><form name="form" method="post" action="http://www.
  <input name="ComboID" type="hidden" id="ComboID" value="1449">
  <input name="ComboName" type="hidden" id="ComboName" value="VC - ATI RADEON 8800GL 128MB DDR Dual Heads w/TV">
  <input name="ComboP" type="hidden" id="ComboP" value="
  $274.85|
  $2.74
  ">
```

home | specials | contact | view cart

Product Catalog Government Sales Corporate Sales

search store

GO

browse store

category  
manufacturer

build a system

barebones  
complete systems

VC - ATI RADEON 8800GL 128MB DDR DUAL HEADS WTV

SKU: 2713159

128 MB  
ATI  
FIREGL 8800

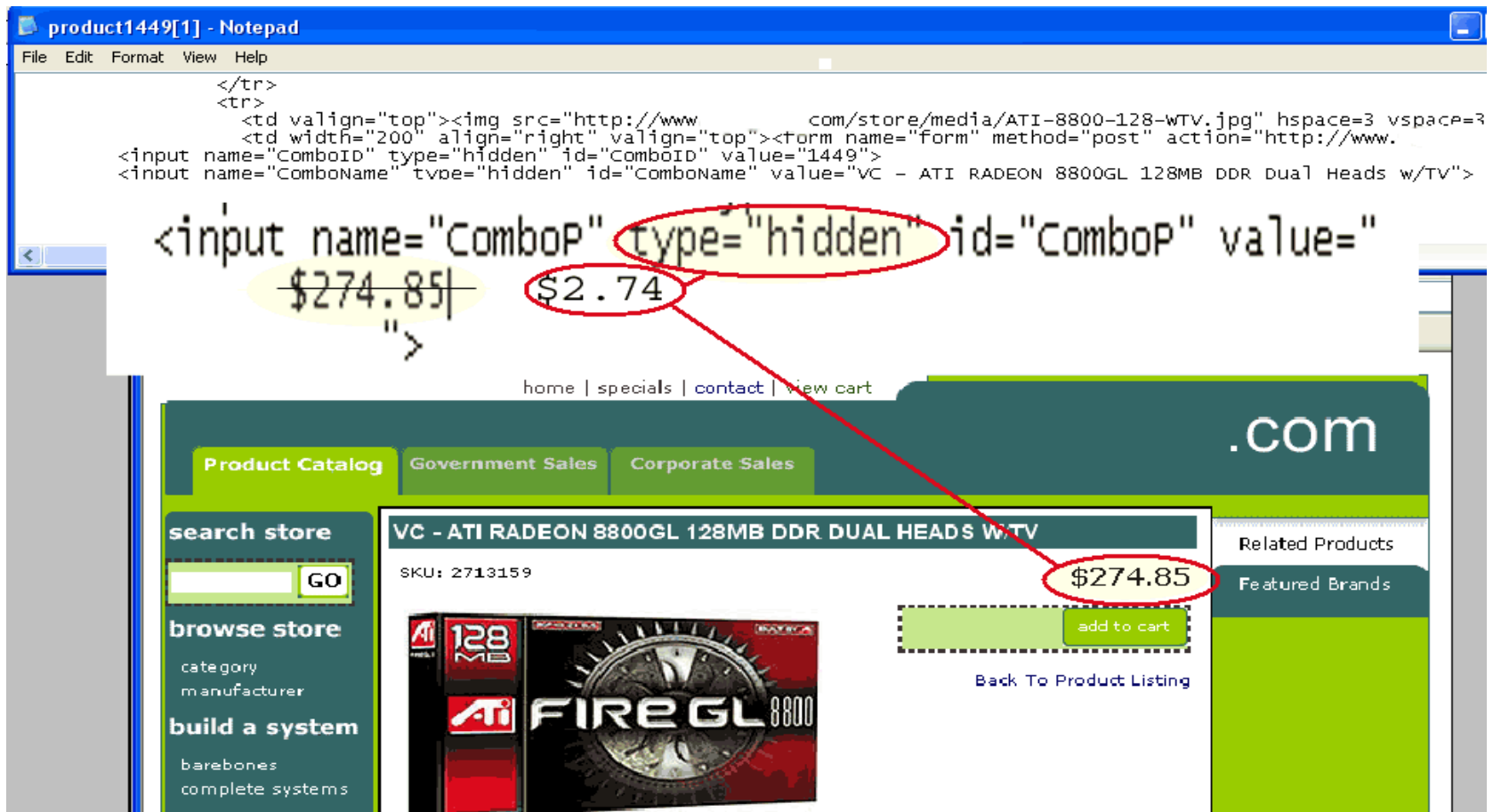
Related Products

Featured Brands

\$274.85

add to cart

Back To Product Listing



# OK! Ora sì che il prezzo è “giusto”!

Google Search Web PageRank 265 blocked AutoFill Options

home | specials | contact | view cart

Product Catalog Government Sales Corporate Sales .com

search store  
GO

browse store  
category  
manufacturer


build a system  
barebones  
complete systems

customer care  
technical support  
returns  
order tracking  
open forum  
terms & conditions  
privacy pledge  
open forum  
terms & conditions

FOLLOWING UPGRADES ARE IMPORTANT FOR YOUR VC - ATI RADEON 8800GL 128MB DDR DUAL HEADS W/TV

Price: \$2.74  
Price (with Selected Options): \$21.12


**Price: \$2.74**

 **Thermal Management**  
Improve Heat Management . For Longer life and to get better Stability.  
Provide yourself with some peace of mind.

- Do not need recommended Heatsink and Fan Solutions
- thermaltake crystal orb for vga card cooling [+\$15.95]
- thermaltake geforce 4 highest performance cooler [+\$22.86]
- thermaltake g4-vga coolmod highest performance cooler [+\$38.82]
- thermaltake g4-vga coolmod highest performance cooler [+\$38.82]

Related Products


ATI RADEON 9800PRO 256MB



Graphics Controller:  
Radeon 9800 Pro  
Memory:  
256MB DDR  
W/TV-out & DVI  
Dual Head

\$484.02 [ info ]

Samsung CD-RW



Samsung CD-RW

# Insicurezze classiche vs. web apps

- Buffer Overflow
  - Eavesdropping
  - Race Condition
  - Man in the Middle
  - Input Validation
  - Session Hijacking
  - Memory Residue
  - Replays
  - Path Manipulation
  - Backdoors
- Buffer Overflow (-)
  - Eavesdropping (+)
  - Man in the Middle (+)
  - Input Validation (+++++)
  - Session Hijacking (++)
  - Replays





# Open Web Application Security Project

## Top Ten Most Critical Web Application Security Vulnerabilities

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Access Control
4. Cross-Site Scripting Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management

Fonte: [www.owasp.org](http://www.owasp.org)



***Mancata validazione degli  
input***

# Validare sempre l'input !

- Abbiamo detto che gli input di una applicazione web sono sempre untrusted
- Tutto quello che arriva va validato:
  - Escape, validate, parse, filter, sanity check
  - TUTTO quello che arriva
  - SEMPRE
- Non esiste un filtro "troppo paranoico", mentre ne esistono migliaia di "non abbastanza paranoici"



# Sequenza di validazione

- Whitelisting: lascia passare solo ciò che riconosci
- Filtering: da questo filtra ciò che, in aggiunta, riconosci come sbagliato
- Escaping: di queste cose, elimina ciò che potrebbe causare confusione
- Parsing: solo ora leggi l'input ed elaboralo

# Validare in positivo !

- Due modi per validare qualcosa
  - Togliere le cose sbagliate (blacklist)
  - Lasciare solo le cose giuste (whitelist)
- Primo principio della sicurezza: *la blacklist è male!*
- *Se una applicazione aspetta, in input, solo BIANCO o NERO, tutte le sfumature di grigio si possono buttare via senza problemi*
  - *E.g.: se mi aspetto un numero, posso buttare via qualsiasi cosa non sia un numero!*



# Filtering, Escaping

- Alcuni caratteri vanno sicuramente rimossi...
  - Ad esempio i caratteri nulli nelle stringhe!
- I caratteri speciali che non possono essere rimossi o bloccati devono essere “escaped” per evitare ambiguità

# Esempio: escaping HTML

- Supponiamo che la nostra web application *non preveda di consentire agli utenti di inserire HTML in un campo... possiamo effettuare il seguente "escaping"*
  - Sostituiamo > con &gt;;
  - Sostituiamo < con &lt;;
  - Sostituiamo " con &quot;;
  - Sostituiamo & con &amp;;

# Altri caratteri malefici

- Altri caratteri a cui prestare attenzione

../ (Directory Transversal)

(\* , ? , +) (globbing)

;" (Append di Comandi)

">" "<" "|" (Data Piping, redirezione di output)

" e ` (Terminatori di stringa)





# E se io voglio l'HTML?

- Se proprio non c'è altra strada si può lasciar passare anche l'HTML, però:
  - Lasciamo passare solo alcuni tag che vogliamo
  - Di questi dovremo filtrare anche gli attributi
- Non è facile capire cosa sia innocuo e cosa no...

# Questi, sicuramente, NO

## ■ Tag:

- <APPLET>
- <BASE>
- <BODY>
- <EMBED>
- <FRAME>
- <FRAMESET>
- <HTML>
- <IFRAME>
- <IMG>
- <LAYER>
- <META>
- <OBJECT>
- <P>
- <SCRIPT>
- <STYLE>

## ■ Attributi:

- STYLE*
- SRC*
- HREF*
- TYPE*

- *E se vi chiedete il perchè, guardate le slide che seguono...*

# Fatta la legge, trovato l'inganno

- Supponiamo di aver correttamente filtrato il tag `<SCRIPT>`:

```
<SCRIPT>alert('JavaScript Executed');</SCRIPT>
```

- E questi altri tag perfettamente equivalenti ?

```
<IMG SRC="javascript:alert('JavaScript Executed');">
```

```
<ANYTHING SRC="javascript:alert('JavaScript Executed');">
```

- Potremmo pensare di fare lo “stripping” dell'attributo SRC e di togliere la parola chiave “javascript:” ...
- Adesso seguitemi nella delirante sequenza di slide che seguono...

# Problema dei whitespace...

- Il mio filtro prende la parola "javascript" e la elimina da SRC... ma che succede se scrivo:

```
<IMG SRC="javasc
```

```
ript>alert('JavaScript Executed');">
```

- Succede che funziona ! :-)
- Soluzione: filtrare i CR-LF, CR, tab, spazi, etc. all'interno dei tag

# Problema delle HTML entities

- Il mio filtro, dopo aver eliminato spazi etc, prende la parola "javascript" e la elimina da SRC. Se ci metto delle entity HTML \09-12 che succede ?  
`<IMG SRC="javasc&#09;ript>alert('JavaScript Executed');">`
- Soluzione: filtrare le entity... ma non è banale!
- E se le metto in esadecimale ?  
`<IMG SRC="javasc&#X0A;ript>alert('JavaScript Executed');">`
- E se ci metto degli zeri davanti ?  
`<IMG SRC=javasc&#000010;ript>alert('JavaScript Executed');>`

# Problema misterioso

- OK. Ora il mio filtro elimina gli spazi bianchi dentro SRC, elimina una serie di regexp per le entity, poi prende la parola "javascript" e la elimina... ma che succede se scrivo:  
`<IMG SRC=" &{alert('JavaScript Executed')} ;">`
- Succede che su alcuni browser funziona ! :-)
- Soluzione: filtriamo via &{

# Problema ricorsivo

- OK. Ora il mio filtro elimina gli spazi bianchi dentro SRC, elimina una serie di regexp per le entity, poi prende la parola "javascript" e i caratteri &{ e li elimina.
- Che succede se scrivo:  
`<IMG  
SRC=" java&{script{alert(' JavaScript  
Executed' ) } ; ">`
- Succede che filtrando via **&{** il resto **sciaguratamente funziona...**
- **Soluzione: non stripping, ma sostituzione**



# Problema maiuscolo

- Ehm... vi siete ricordati che tutti questi filtri devono essere anche case-insensitive, vero ?



# Altro esempio: style sheet

- Cambiamo tag, prendiamo “STYLE”

```
<style TYPE="text/javascript">JS EXPRESSION</style>
```

- Anche qui dobbiamo strappare via “javascript”, come prima... ma è sufficiente ?

- Considerate anche questi:

```
<STYLE type=text/css>  
@import url(http://server/very_bad.css);  
@import url(javascript:alert('JavaScript  
Executed'));  
</STYLE>
```

- Filtriamo via @import ! E vai che si riparte !

# Style è anche un attributo...

- E puntualmente il javascript funziona pure da lì dentro

```
<P STYLE="left:expression(eval('alert('\ JavaScript Executed\ ');window.close()'))" >
```

- Questo è un bel dramma. O filtriamo via l'attributo STYLE tout-court, o escogitare un modo per filtrarlo è drammatico... dovremmo partire eliminando “left:”, “expression” ed “eval”... e chissà che altro!



# Concludendo sulla validazione

- Filtrare gli input. Filtrarli sempre. Filtrarli tutti.
- Filtrare accettando ciò che è buono, non scartando ciò che è cattivo
- HTML è cattivo, molto più cattivo di quanto non si pensi a prima vista
- Se state iniziando a pensare ai web services con un po' più di orrore... fate bene!



# ***Insecure Storage***

# Cookie Poisoning

- Come abbiamo detto, HTTP è stateless (accidenti!)
- HTTP è tendenzialmente unidirezionale (il server non può “leggere” nulla dal client)
  - Ad eccezione di get e post, chiaramente!
- Meccanismo dei “cookie” per consentire al server di salvare e ricaricare informazioni “lato utente”
  - Idea originale: personalizzazione dei siti
  - Abuso: raccolta dati sui navigatori
  - Idea tendenzialmente pericolosa: uso per l'autenticazione

# Cookie Poisoning

Welcome - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites Media

Address <https://www.bankcard4me.com> Go Links >>

Google Search Web PageRank 269 blocked AutoFill Options

## BankCard4Me.com

Transaction Activity Payment Information Account Management

**MY PROFILE WELCOME HELP SIGN OFF**

WELCOME, **Mr. Newell** Account: xxxx-xxxx-xxxx-6000

[Click here](#) to view your account information.

[Transfer balances](#) from high rate cards!


Apply for a higher [credit limit](#).


Find out about great [promotions](#) from Commerce Bank!


Learn about [Automatic Bill Pay!](#)

Dispute an [unauthorized transaction](#).

### Product Information

Select   
**Special Connections<sup>SM</sup> Visa**  
Classic/Gold/Platinum

Select   
**Commerce Miles Visa<sup>®</sup> Gold**

Select   
**Royals<sup>®</sup> MasterCard<sup>®</sup>**  
Classic/Platinum

# Cookie Poisoning

The screenshot shows a Microsoft Internet Explorer browser window with the address bar displaying `https://www.bankcard4me.com`. The browser's title bar reads "Welcome - Microsoft Internet Explorer". The website content is partially obscured by a Notepad window titled "Cookie.bankcard1 - Notepad".

The Notepad window contains the following text:

```
User ID  
mdvdkk  
www.bankcard4me.com  
1024  
2140300160  
29691790
```

To the right of the Notepad window, the text "UserID (mdvdkk) newell" is displayed, indicating that the cookie's value has been changed from "mdvdkk" to "newell".

The website page shows a "WELCOME, Mr. Newell" message and a navigation menu with links for "Transaction Activity", "Payment Information", and "Account Management". The account number is displayed as "Account: xxxx-xxxx-xxxx-6000".

Below the welcome message, there are several links and promotional text:

- [Click here](#) to view your account information.
- Find out about great [promotions](#) from Commerce Bank!
- [Transfer balances](#) from high rate cards!
- Learn about [Automatic Bill Pay!](#)
- Dispute an [unauthorized transaction](#).
- Apply for a higher [credit limit](#).

On the right side of the page, there is a "Product Information" section with three items:

- [Select](#) **Special Connections** Classic/Gold/Platinum
- [Select](#) **Commerce Miles Visa® Gold**
- [Select](#) [Image of a Visa card]

# Cookie Poisoning

Welcome - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <https://www.bankcard4me.com>

Google Search Web PageRank

**BankCard4Me.com**

Transaction Activity Payment Information Account Management

**Cookie.bankcard1 - Notepad**

```
File Edit Format View Help
UserID
mvdvkk
www.bankcard4me.com
1024
2140300160
29691790
```

**Changed To UserID (fzqyz)**

MY PROFILE WELCOME HELP SIGN OFF

Account: xxxx-xxxx-xxxx-6000

WELCOME, **Mr. Garza**

[Click here](#) to view your account information.


Find out about great [promotions](#) from Commerce Bank!

[Transfer balances](#) from high rate cards!


Learn about [Automatic Bill Pay!](#)

Dispute an [unauthorized transaction](#).


**Product Information**

Select 

**Special Connections**  
Classic/Gold/Platinum

Select 

**Commerce Miles Visa® Gold**

Select 





# Altri esempi di cookie poisoning

- Modificare i cookie estendendone l'expire time per evitare il logout automatico
- Se i cookie contengono degli identificativi generati in modo poco sicuro possono essere indovinati



# ***Controllo d'accesso e autenticazione***



# Cosa fare con le password

- Le password sono un dato da proteggere
  - Non devono essere salvate in plaintext
  - Devono essere inserite in SSL
  - Non devono durare in eterno
- Devono sottostare ad alcuni vincoli, es.
  - Lunghezza di almeno 8 lettere
  - Non contenere il nome utente o un pezzo di nome utente
  - Non compaiono in un dizionario
  - Almeno un numero e un carattere speciale

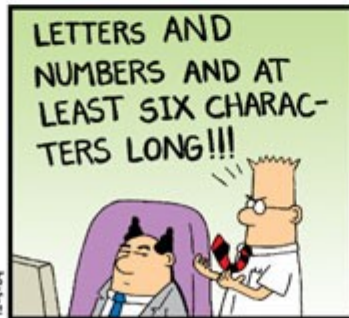
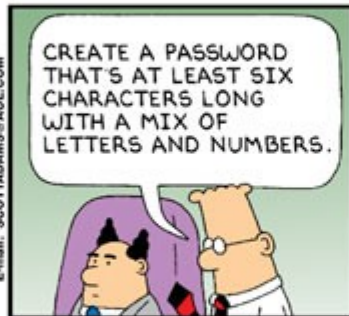
# Cose da non fare mai

- Mettere una lunghezza massima
- Mettere un sistema per cambiare la password in qualcosa di prestabilito (password di “emergenza”)
- Inviare la password in chiaro, non su SSL (ad esempio in una e-mail...)
- ... chiedere password **troppo complicate rispetto alla mentalità degli utenti**



# DILBERT<sup>®</sup>

BY  
SCOTT ADAMS



# Attacchi di brute force

- Brute force (definizione banale): “tentare di indovinare le password”
- Soluzione banale: dopo *n tentativi di login falliti su un account, blocchiamo l'account e l'indirizzo IP di provenienza. Funziona ?*

# Reverse brute force

- Bene: allora fissiamo una password o un set di password minore di  $n$  e *proviamole a turno su tutti gli account !*
  - Successo veramente su applicazioni bancarie
- Soluzioni:
  - Rendere account non enumerabili
  - Bloccare un indirizzo IP dopo un certo numero di tentativi

# Dal brute force al DoS

- Adesso, se un aggressore vuole chiudere fuori dal sito un utente, deve solo fare una sequenza di tentativi di login falliti. Bel risultato!
- Soluzione parziale: segnare l'ip insieme al blocco dell'account. Lasciare fare almeno un tentativo a un ip diverso
- Che succede con l'uso di grandi proxy stile AOL o reti con NAT stile FastWeb ? Brrr.



# Autenticazione mediante cookie

- Mescoliamo due cose critiche = otteniamo una cosa esplosiva
- **Non salvate le credenziali nei cookie (possono essere sottratti)**
- **Fate in modo che i cookie non possano essere riutilizzati (possono essere sottratti)**
- **Non usate la durata dei cookie per forzare il logout (può essere manipolata)**



# Non-riuso dei cookie

- Collegate il cookie all'indirizzo IP (inserirlo nell'hash ad esempio)
- Collegate il cookie all'identificazione HTTPClient negli header, a -User-Agent, ad -Accept-Language
- Riautenticazione con password per funzioni critiche
- Uso del referer come “guardia” (NON è affidabile se è giusto, ma se è sbagliato...)



# Risorsa utile da meditare

- Kevin Fu, Emil Sit, Kendra Smith, e Nick Feamster: “Do's and Don'ts of Client Authentication on the Web”  
<http://cookies.lcs.mit.edu/pubs/webauth.html>



# ***XSS: Cross Site Scripting***

# Cross-Site Scripting (XSS)

- ***Definizione: iniezione di codice di scripting su una pagina da parte di un esterno***
- ***Gravità:***
  - Tramite social engineering possiamo far eseguire Javascript agli utenti
  - Cookie theft
  - Session hijack
  - Esecuzione di transazione fraudolente
  - Accesso a informazioni riservate

# Esempio di XSS

- Una form web di feedback che non filtra appropriatamente JavaScript e ri-visualizza il commento all'utente
- Un malintenzionato può
  - Studiare come chiamare la form tramite un URL
  - Costruire un Javascript che legge il cookie dell'utente e lo invia a un sito differente
  - Encodare il JavaScript in esadecimale, così sembra solo un URL qualsiasi
  - Alternativamente, potrebbe fare un popup che chieda login e password e farseli mandare via e-mail



# XSS – Come risolviamo

- Validare l'input, come abbiamo già detto, in particolare bandire lo scripting
- Possibilmente, usiamo dei framework esistenti che si siano dimostrati storicamente “ben scritti”



# ***Gestione degli errori***



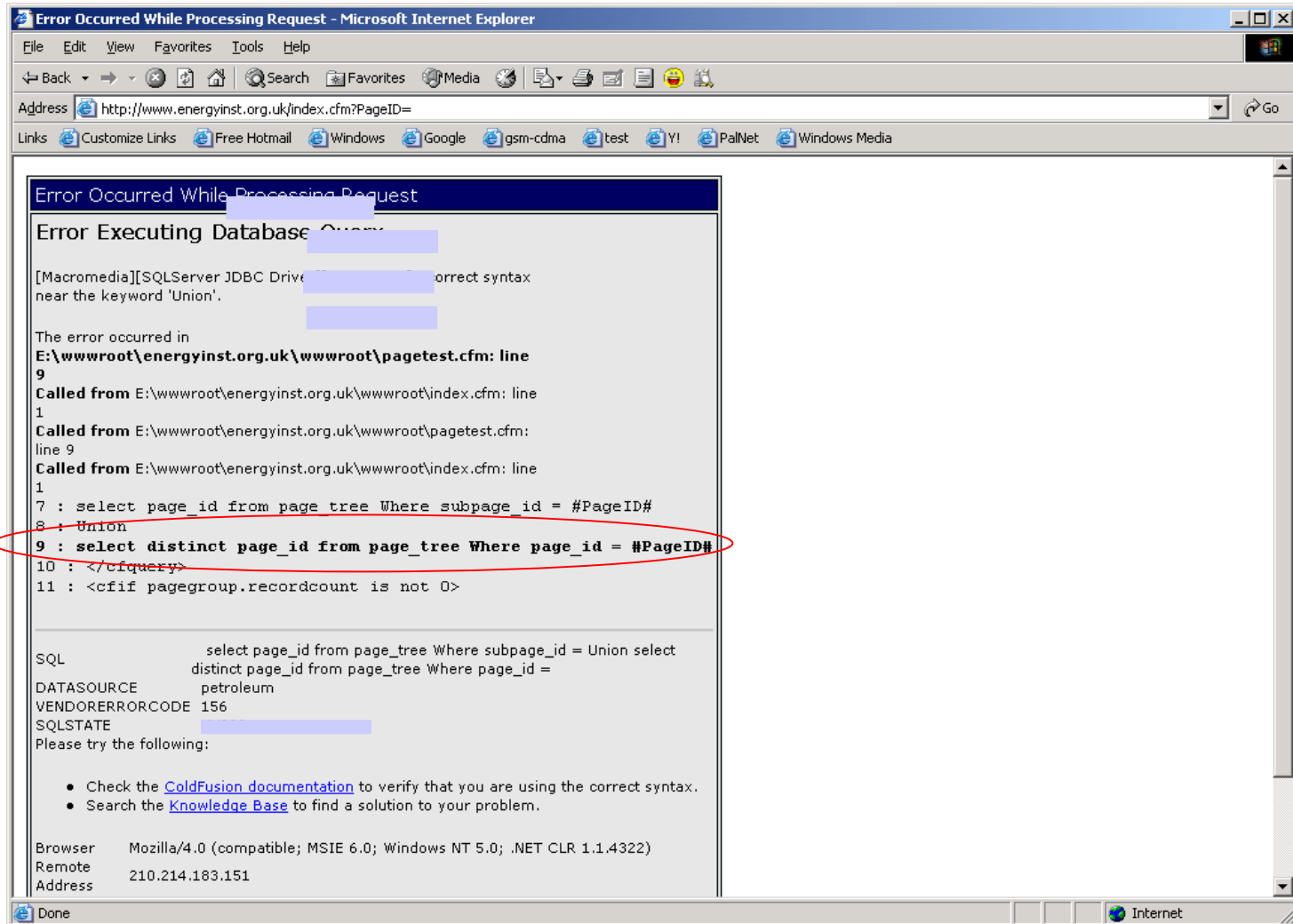
# Gli errori degli errori

- Un simpatico e servizievole messaggio d'errore fa sempre piacere
- Tuttavia, è proprio necessario che esso faccia l'echo di cose inserite dall'utente ?
  - No: allora non facciamolo
  - Sì: allora dobbiamo filtrare, strappare e validare
- Ancora: non siamo troppo specifici nei messaggi (e.g. “hai sbagliato la password”)

# Errori rivelatori (freudiani ? :)

- Per default un server stampa messaggi d'errore di debug. Questo va bene, appunto, durante il debug
- In produzione sostituite i messaggi completi con dei gentili “Ci dispiace, non ha funzionato, riprova”, e lasciatela lì
- Soprattutto, eliminate messaggi che rivelino:
  - Versioni di server
  - Nomi di database
  - Path

# Esempio di enumerazione



**Error Occurred While Processing Request**

**Error Executing Database Query**

[Macromedia][SQLServer JDBC Driver] Incorrect syntax near the keyword 'Union'.

The error occurred in  
E:\wwwroot\energyinst.org.uk\wwwroot\pagetest.cfm: line 9  
Called from E:\wwwroot\energyinst.org.uk\wwwroot\index.cfm: line 1  
Called from E:\wwwroot\energyinst.org.uk\wwwroot\pagetest.cfm: line 9  
Called from E:\wwwroot\energyinst.org.uk\wwwroot\index.cfm: line 1

```
7 : select page_id from page_tree Where subpage_id = #PageID#  
8 : Union  
9 : select distinct page_id from page_tree Where page_id = #PageID#  
10 : </cfquery>  
11 : <cfif pagegroup.recordcount is not 0>
```

SQL select page\_id from page\_tree Where subpage\_id = Union select distinct page\_id from page\_tree Where page\_id =  
DATASOURCE petroleum  
VENDORERRORCODE 156  
SQLSTATE  
Please try the following:

- Check the [ColdFusion documentation](#) to verify that you are using the correct syntax.
- Search the [Knowledge Base](#) to find a solution to your problem.

Browser Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)  
Remote Address 210.214.183.151

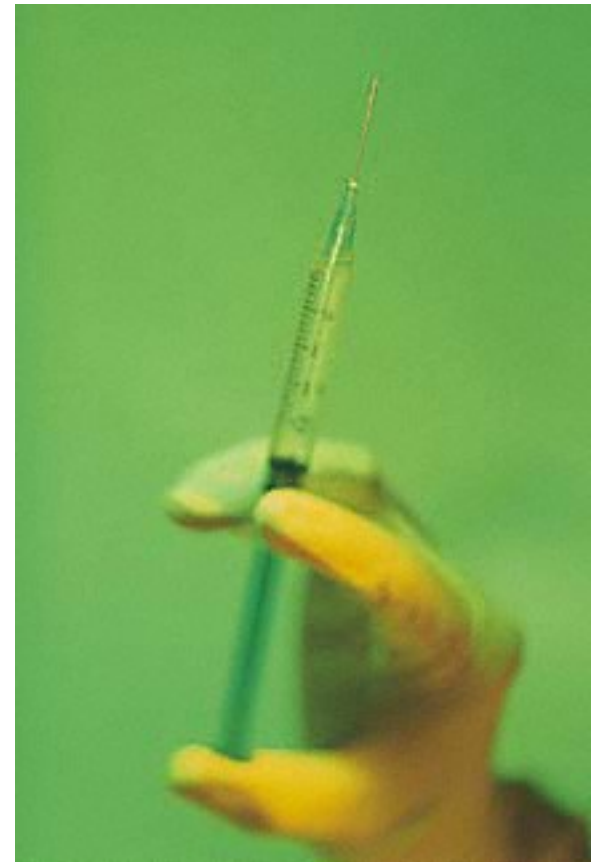
Done Internet



# ***Injection flaws***

# SQL Injection

- Si definisce “SQL Injection” un errore che consenta a un aggressore di far “filtrare” comandi SQL verso un database
- Il problema avviene quando un input dell'utente viene usato per costruire una query SQL
- A seconda dei privilegi della query “iniettata” l'aggressore può fare più o meno qualsiasi cosa...





# SQL Injection – Esempi tipici

- Codice per motori di ricerca: costruiscono le query dall'input dell'utente
- Query di login fatte mediante catenazione di stringhe
- INSERT costruite da form che replicano la struttura del DB

# Ma non lo scopriranno mai...

- Pensiamo a una form di ricerca
- Ci inseriamo “tom's house” e otteniamo un errore... ci si può arrivare anche per caso!
- Chiunque voglia attaccare una web app proverà con apostrofi e altri caratteri di iniezione!
- Se poi come spesso accade i campi della web app hanno gli stessi nomi delle colonne del DB e con qualche messaggio di errore si può fare enumeration...
- Insiders, anyone ?

# SQL Injection - Esempio

```
public void OnLogin(object src, EventArgs e){
    SqlConnection con = new SqlConnection(
        "server=(local);database=myDB;uid=sa;pwd;" );

    string query = String.Format(
        "SELECT COUNT(*) FROM Users WHERE " +
        "username='{0}' AND password='{1}'",
        txtUser.Text, txtPassword.Text );
    SqlCommand cmd = new SqlCommand(query, con);
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    try{
        if(reader.HasRows())
            IssueAuthenticationTicket();
        else
            TryAgain();
    }
    finally{
        con.Close()
    }
}
```



# SQL Injection – Spiegazione

## Cosa pensava il programmatore:

username: abc  
password: test123

La SQL query risultante diventa:

```
select * from users where username='abc' and password =  
'test123'
```

## Cosa NON pensava il programmatore:

username: abc'; --  
password:

La query risultante diventa:

```
select * from users where uname='abc'; --' and password=''
```

# SQL Injection – The Problem

## Cosa pensava il programmatore:

Username: doug

Password: p@\$w0rd

```
SELECT COUNT(*)
```

```
FROM Users
```

```
WHERE username='doug' and password='p@$w0rd'
```

## Cosa CONTINUAVA a non pensare:

Username: ' OR 1=1 --

Password:

```
SELECT COUNT(*)
```

```
FROM Users
```

```
WHERE username='' OR 1=1 -- and password=''
```

# Prevenire le iniezioni

- ***Usiamo dei PreparedStatements***
- ***Validiamo tutto quel che entra***
- ***Lasciamo passare SOLO ciò che può servire e IN OGNI CASO togliamo i caratteri speciali ' o --!***
- ***Eliminiamo gli errori SQL (enumerazione)***
- ***Non usiamo i nomi dei campi del DB come nomi dei campi delle form***
- ***Limitiamo i privilegi degli utenti con cui vengono eseguite le varie query***





# ***URL manipulation***

# URL Manipulation

The screenshot shows a Netscape browser window titled "Pharmacy - Netscape". The address bar contains the URL: `http://www.abc.com/pharmacy/pre.asp?back=/pharmacy/scripts.asp&patientid=790865`. A green circle highlights the `patientid=790865` portion of the URL, with a green arrow pointing to a green box containing the text `patientid=790865`. The website content includes a navigation menu with "home", "health", "beauty", "wellness", "personal care", and "pharmacy" (highlighted in red). Below the menu is a red banner with the text "Prescriptions and refills delivered to your door." and a navigation bar with "your list", "shopping bag", "checkout", "your account", "prescriptions", and "help". The main content area displays "pharmacy | Health Profile" for "Jenny Smith" with an "Update Profile" link. Personal information includes: Sex: Female, Birthday: 5/5/1970, Phone number: 408-4345756, Address: 343 1st st, San Jose, CA, Medical Conditions: Pregnancy ; AIDS, and Current Medication: Prozac. A footer navigation bar contains "your list", "shopping bag", "checkout", "your account", and "help".

# URL Manipulation

The screenshot shows a Netscape browser window titled "Pharmacy - Netscape". The address bar contains the URL: `http://www.abc.com/pharmacy/pre.asp?back=/pharmacy/scripts.asp&patientid=*`. A green circle highlights the `patientid=*` portion of the URL. A green arrow points from this circle to a green box containing the text `patientid=*`. The website content includes a navigation menu with buttons for "home", "health", "beauty", "wellness", "personal care", and "pharmacy". Below the menu is a red banner with the text "Prescriptions and refills delivered to your door." and a secondary menu with buttons for "your list", "shopping bag", "checkout", "your account", "prescriptions", and "help". The main content area displays two patient profiles:

**Abare Kelly** [Update Profile](#)

Sex: Female  
Birthday: 8/4/1965  
Phone number: 256-5457674  
Address: 434 South st, Atlanta, Georgia  
Medical Conditions: Asthma ; High Blood Pressure  
Current Medication: Ambien

**Abba Kevin** [Update Profile](#)

Sex: Male  
Birthday: 7/3/50  
Phone number: 334-5432346  
Address : 434 Concord Dr, Pheonix City, AL  
Medical Conditions: Cancer

The browser's status bar at the bottom shows "Document: Done".



# URL Manipulation

- Una richiesta GET manda parametri anche critici sulla URL... sono visibili e facilmente manipolabili
- Una POST complica lievemente la cosa, ma non è che sia una grande protezione
- Le GET rimangono però nella history, che è peggio...
- Come per gli hidden value e per i cookie... non fidarsi del client ! Mai !



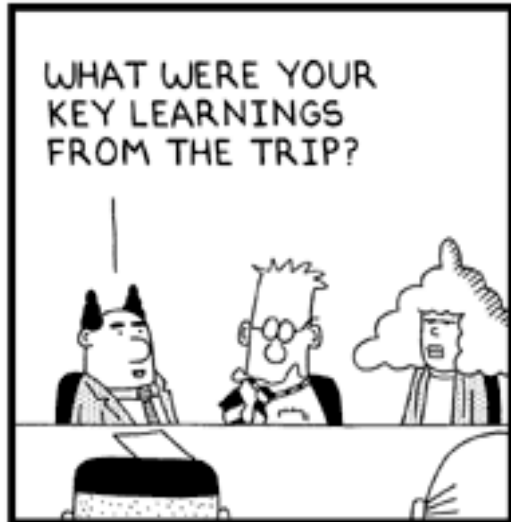
# URL Manipulation - Soluzione

- Evitare di usare le GET
- Validare i parametri con i token di sessione
- Possibilmente proteggere i parametri passati tra client e server, magari usando degli hash in modo intelligente
- Se dobbiamo salvare token sul client, firmiamoli digitalmente in qualche modo





# ***Session hijacking***



www.dilbert.com scottadams@aol.com

1-30-04 © 2004 Scott Adams, Inc./Dist. by UFS, Inc.

# Session hijacking-Problema

- Normalmente fare hijacking richiede l'intercettazione e la manipolazione del traffico di rete (e.g. MITM)
- HTTP è stateless (ripetiamoci), quindi le “sessioni” possono essere violate molto più facilmente, ad esempio
  - Sottraendo un cookie con un attacco XSS
  - Facendo brute forcing sui valori degli id
  - Oppure, perché no, con un po' di sniffing

# Session hijacking: difendersi

- Utilizzare HTTPS, per tutta l'applicazione o almeno per i passaggi critici
- Far corrispondere il cookie all'indirizzo IP sorgente
- Usare ID grandi per un brute force attack
- Usare un session-id cookie che cambia a ogni richiesta in modo controllabile, così che l'utente a cui viene sottratto un cookie si trovi “bloccato” e percepisca il problema



# ***XML e SOAP: Web Services***



# XML nelle pagine web

- XML è un metalinguaggio per costruire linguaggi che specificano dati
- Spesso usato “semplicemente” per salvare dati eterogenei dietro alle web application
- Un modo flessibile per fare siti dinamici
- Se viene accettato input dall'utente, un dramma...
- Esempio: sito di ricette, anche inserite dagli utenti

```
<?xml version="1.0"?>
```

```
<RECIPEBOOK>
```

```
  <RECIPE NAME="Spaghetti" SERVES="2">
```

```
    <INGREDIENTS>
```

```
      <INGREDIENT>Angel hair pasta</INGREDIENT>
```

```
      <INGREDIENT>Pasta sauce</INGREDIENT>
```

```
    </INGREDIENTS>
```

```
    <DIRECTIONS>
```

```
      Boil large pot of water. Add pasta. Mix with sauce.
```

```
    </DIRECTIONS>
```

```
  </RECIPE>
```

```
  <RECIPE NAME="Garlic Bread" SERVES="4">
```

```
    <INGREDIENTS>
```

```
      <INGREDIENT>French bread</INGREDIENT>
```

```
      <INGREDIENT>1 Garlic clove</INGREDIENT>
```

```
      <INGREDIENT>Butter</INGREDIENT>
```

```
    </INGREDIENTS>
```

```
    <DIRECTIONS>
```

```
      Cut loaf in half. Mince garlic and add to butter.
```

```
      Butter both halves. Pre-heat oven to 400.
```

```
      Bake loaves for 5 mins.
```

```
    </DIRECTIONS>
```

```
  </RECIPE>
```

```
</RECIPEBOOK>
```

# XML injection

- Se nel commento viene inserito dell'XML...
  - Stessi problemi di SQL Injection + Cross-site scripting + HTML malformato, on steroids!
- Le specifiche di XML consentono di usare tag che richiamano applicazioni server-side
  - Aaaaaaaaaaaaaaaaaaaaaaargh !
- Filtra e valida: “Sempre più difficile, siore e siori!”
- Utilizzo del DTD per validare i documenti





# Web services

- RPC (Remote Procedure Call)
- Over XML (eXtensible Markup Language)
- Over SOAP (Simple Object Access Protoc.)
- Over HTTP
- = niente di nuovo sotto il sole, o quasi...
- ... ma MOLTO di nuovo per la security !



# What a wonderful world!

- I web services sono un metodo per consentire alle applicazioni di trasmettersi dati in formati universali, per consentire la composizione di oggetti distribuiti
- Studiata apposta per passare attraverso i firewall (“usiamo la porta 80, almeno non la possono chiudere!”) - Grande idea !



# ***Programmare applicazioni sicure***

# Riassumendo i consigli chiave

- ***Trust no one***
  - *Specialmente NON il client*
  - *Validate sempre, tutto, bene, con whitelist*
- ***Authenticate sempre, authenticate tutto***
  - *Usate SSL per le operazioni critiche e il login*
  - *Richiedete la password per le operazioni più critiche*
  - *Collegate le sessioni all'IP*
  - *Mai autenticazioni o controlli client-side*
- ***Usate algoritmi crittografici solidi e standard***

# Riassumendo i consigli chiave

- ***Possibilmente, non visualizzate input dell'utente***
  - Se non è necessario, evitate input in HTML o XML
  - Se proprio è necessario, accettate solo un sottoinsieme molto ridotto
- **Non visualizzate messaggi d'errore o di debug...**
  - E in ogni caso, non metteteci quelli di default
  - Non metteteci i nomi di database e simili
  - **NON METTETECCI L'INPUT DELL'UTENTE**

# Code security review

- ***Un processo di quality assurance, interno o esterno, troppo spesso trascurato***
  - ***“L'importante è che sia online, poi ci pensiamo”***
- ***Effettuare una code review previene dei rischi gravissimi, anche civili e penali!***
- ***Esistono dei tool automatizzati, ma hanno un costo considerevole ed efficacia spesso limitata***
- ***A volte non c'e' un esperto di security in azienda, a volte se c'è non è anche un esperto***



# Assessment Esterno

- Aumenta il numero di occhi che guardano il codice incrementando la sicurezza
- Spesso economico rispetto a training + esperienza + costo dei tool
- Trasferimento di conoscenza dall'esperto agli sviluppatori sui metodi per costruire applicazioni sicure