Per Principianti

Difesa delle Web Application

Nanni Bassetti

Grado di difficoltà

La complessità e la sicurezza nelle Web Application rappresentano un binomio ideale per addestrarsi alla difesa da coloro che vogliono ficcare il naso lì dove non dovrebbero.

Proprio per offrire all'utente navigatore una serie di servizi utili ed interattivi, le web application lasciano aperti parecchi fronti di attacco, vettori di input malevolo, che permettono, se non adeguatamente protetti, esporre informazioni riservate o mandare in crash l'intero sito, queste considerazioni fanno aumentare la complessità di sviluppo di un sofware online, che non deve essere solo efficiente, efficace ed usabile, ma anche sicuro.

Ci sono molte metriche per valutare l'importanza di un Web site, per esempio, un'applicazione può avere importanza critica basata sulla relativa generazione del reddito, sulla criticità ad un processo di affari, o sulla sensibilità dei dati che gestisce.

Ciascuno di questi punti fornisce le risposte necessarie per dare la priorità alla sicurezza implementata su di un'applicazione.

Tra le migliori filosofie di protezione c'è l'oscurare le proprie risorse il più possibile, questo è tratto dall'*Arte della guerra* di Sun Tsu, quindi un concetto che deriva dal mondo militare, che ci invita a non mostrare più del necessario, per poi sorprendere l'avversario.

Traslando questo concetto nel mondo dell'informatica, si traduce in concetti di *information hiding*, che vanno dal non dare nomi immaginabili ai file o alle directories, alle variabili nei form html e nei campi dei database, ecc. Ecc.

Per esempio se abbiamo una cartella nella quale l'applicazione web conserva dei documenti PDF, è meglio chimarla: *lert900iok* invece che *pdf* ed il software che permetterà il download dei documenti contenuti in essa, non deve mostrare il percorso in querystring o in post, ma si utilizzino tecniche come l'*adodb.stream* dell' ASP 3.0, che permette di creare dei link di download del tipo:

```
<a href="download.asp?file=xxx.pdf">
Download XXX.PDF</a>
```

Dall'articolo imparerai...

- A creare web application sicure a prescindere dal linguaggio di programmazione usato

Cosa dovresti sapere...

- Concetti di amministrazione di sistema e programmazione web application

Anche nei form html bisogna evitare di usare i nomi dei campi <*input*> che coincidano coi nomi dei campi del database al quale invieranno i loro valori al fine di generare query.

```
<input type="hidden" name=
"ut" value="ciccio">
```

mentre il campo del database si chiamerà "username" così si crea un ulteriore ostacolo all'attaccante.

Chiaramente la *sicurezza attraverso l'oscurità* (nascondere le informazioni) non è una panacea, ma solo una serie di ostacoli in più da porre sul cammino di chi tenta di scardinare la nostra web application.

Enumeriamo i componenti dell'applicazione

Attraverso l'enumerazione dei componenti di un software online possiamo avere anche una misura dell'esposizione agli attacchi dello stesso:

I parametri URL: molte applicazioni usano parecchi parametri in URL o querystring, che servono a tener traccia di sessioni utente o shopping cart, sono molto utili, ma espongono parecchio il software e danno informazioni, meglio criptarli, anche con algoritmi self-made. Inoltre in questi parametri si possono inserire i famigerati XSS (Cross Site Scripting), ossia degli script (es. in Javascript) che modificano il funzionamento dell'applicazione o addirittura dirottano l'utente verso siti cloni del sito originale, che servono solo a carpire i dati dell'ignaro navigante (vedi fenomeno Phishing).

HTML forms: I forms contengono svariate variabili e informazioni sul file di arrivo di quest'ultime, così da poter tentare delle adulterazioni. Un esempio classico è quello dello shopping cart realizzato così:

```
<form action="ordina_ora.asp"
method="post">
<input type="hidden" name="
nome_articolo" value="Acer 1690">
<input type="hidden" name=
"prezzo" value="1000">
<input type="submit" value=
"PAGA ORA">
</form>
```

Salviamo la pagina HTML sul nostro computer, poi la editiamo e sostituiamo l'*action* del form con *action=http://www.negozio-online-farlocco.com/carrello/ordina.asp* e adulteriamo il *value* del campo prezzo:

```
<input type="hidden" name=
"prezzo" value="600">
```

Ed ora clicchiamo su *PAGA ORA* Bene abbiamo appena risparmiato 400 euro.

Uso dei cookies: anche i cookies devono essere usati con parsimonia ed oculatezza perché, come per i parametri in URL possono dare molte informazioni di user tracking e sessione.

Gli headers delle pagine: evitare di scrivere troppe cose negli header delle pagine in modo da dare informazioni sul server o sul software usato per programmare, ecc...

Bad response codes: HTTP response codes come il 404 o il 500 possono essere usati per carpire informazioni sui percorsi dei files, le variabili di database o il tipo di server/database, quindi impostare sul web server delle pagine ad hoc che rispondano al 404 o al 500.

Directory di database storage: impostare il web server di non servire la cartella contenente i database es: *mdb-database*, che deve avere i diritti di lettura e scrittura da parte del sistema e dell'utente IUSR ma non esser raggiungibile dal web server così da evitare download di file MDB (se si usa MS ACCESS). In ogni caso nominare i database coi nomi più strambi per scongiurare il *name guessing,* ossia quella lotteria di nomefile, che l'hacker tenta per indovinare il nome giusto ed iniziare il download (Es: database.mdb)

SQL Injection: Quando si inviano dati, come la username e la password, tramite un form html, queste due variabili finiscono in uno script (asp, php, perl, ecc.), che li passa ad una query SQL (il linguaggio di gestione del database). Se nei campi sudetti inseriamo delle stringhe SQL come:

```
' or user like '%

' or password like '%
```

Così potremmo spezzare la stringa di SELECT sql usata nello script ed entrare nel sistema come primo utente:

```
select * from utenti where user like
'"&user&"' and password like '"
&password&"'"
diventa select * from utenti where
user like '' or user like '%' and
password like '' or password like '%'
```

dove il simbolo % è un simbolo jolly che significa *tutti*. Chiaramente di SQL Injection se ne possono fare tante, alcune servono a capire che tipo di database un sito usa, altre a capire i nomi dei campi del database, esempio: *http://www.negozio-online-farlocco.com/prodotto.asp?id=123*

Lo avveleniamo con: *http://www.negozio-online-farlocco.com/prodotto.asp?id='* che fornirà un messaggio di errore molto esplicativo.

Per contrastare le SQL Injection conviene rimpiazzare tutti i caratteri delimitatori del dialetto SQL del database usato con caratteri vuoti ed usare sempre istruzioni di intercettazione dell'errore in modo da non visualizzare sul browser messaggi di errore indicativi.

Applet Java: Evitare di usare le

Cenni sull'autore

L'autore lavora dal 1998 nella IBOL S.r.l. come sviluppatore di applicazioni web e system administrator e dal 2004 si occupa di sicurezza informatica con la sua ditta NBS*http://www.nannibassetti.com* . Ha scritto un libro sulla sicurezza del web ed un thriller informatico "Onphalos" ed ha collaborato con la procura del Tribunale di Bari per un indagine informatica su un computer sequestrato (Digital Forensics).

applet java per proteggere aree di accesso del sito, visto che sono facilmente decompilabili con software come il *NMIJavaCodeViewer.*

ADMIN AREA: Per concludere l'area di amministrazione è meglio proteggerla con l'accesso consentito ad un utente impostato sul sistema, (la famosa finestra grigia che appare in pop-up) è il metodo più sicuro.

Contiamo l'ESPOSIZIONE

Una volta enumerati i fattori di esposizione di una web application, possiamo avere una quantificazione aritmetica del rischio alla quale essa è esposta.

Creare un exposure rating aiuta i programmatori a decidere sugli sforzi e sulle contromisure da adottare Il rating rappresenta gli aspetti fondamentali di un sito web, come i forms, i cookies ed i parametri URL, col vantaggio di non effettuare test intrusivi o pericolosi, che sono spesso fonte di nervosismo, bensì basteranno dei test condotti usando un semplice web browser così da valutare velocemente i punti di debolezza dell'applicazione.

Come si genera un exposure rating? Semplicemente contando tutti i fattori di esposizione, esempio:

se abbiamo un form, dei cookies e dei parametri URL possiamo dire che ci sono 3 fattori di esposizione, così il nostro exposure rating è pari a 3.

Molti potrebbero pensare a dare dei pesi ai fattori espositivi, così da rendere più importanti alcuni e meno altri, ma questo ingenera soltanto false sicurezze.

L'esposizione ha il vantaggio di poter essere quantificata all'inizio della progettazione dell'applicazione ma il rating può essere prodotto anche per applicazioni già realizzate, rilasciando delle patch di codice, riutilizzabili per lo sviluppo di altre web apps future.

Conclusioni

Per concludere possiamo sintetizzare in questo modo:

Dobbiamo identificare sempre i punti deboli di un software online prima di iniziare a scriverlo.

E' necessario evitare le vulnerabilità ben note.

Si deve immaginare di complicare al massimo l'accesso alle informazioni riservate.

Dovete immaginare sempre come un hacker potrebbe rubare dei dati e quindi aggiungere la vostra intuizione ai fattori di esposizione.

Una volta blindato il sistema, sarà veramente difficile poter avere accesso al vostro patrimonio informativo, magari riusciranno a mandare in crash il server, a fare degli attacchi DOS, ma non avranno il vostro scalpo.