

PHP E SICUREZZA NELLE PAGINE WEB (php anti acaro)
evilsocket
<http://evilsocket.altervista.org/>

:: Introduzione

In questo paper andremo ad analizzare alcuni aspetti fondamentali della sicurezza delle pagine in php, di conseguenza gli attacchi più noti e le tecniche di base per "blidare" il proprio codice dagli stessi .

:: Concetto

Fondamentalmente, ogni tipo di attacco sfrutta le informazioni che vanno dal pc dell'attaccante alla pagina stessa, che siano parametri POSTATI da un form, o parametri mandati via url in modalità GET .

Esempio di una richiesta GET :

```
httpwww.sito.com/pagina.php?campo=1234
```

Ed in questo caso il parametro "campo" sarà contenuto nella variabile speciale **\$_GET** del PHP .

Esempio di una richiesta POST :

```
<form action='pagina.php' method='POST'>
  <input type='hidden' name='campo' value='1234'>
</form>
```

Dove il parametro "campo" verrà postato dal form a pagina.php che lo riceverà all'interno della variabile php **\$_POST** .

In entrambi i casi mi sembra evidente l'importanza di un "filtro", messo prima dell'elaborazione stessa dei dati, che controlli questi campi e li ripulisca dall'eventuale merda sql o xss che il furbacchione di turno ci ha infilato .

:: Gli attacchi

Prendiamo ad esempio il primo caso (la richiesta GET) e vediamo come un hacker potrebbe sfruttare la situazione per una sql injection :

```
http://www.sito.com/pagina.php?campo=1234 and 1=2 union all select
table_name,1,2 ... N from information_schema.tables/*
```

In questo caso, tramite una sql injection, vengono selezionati i nomi di tutte le tabelle presenti nel db, per una successiva analisi del loro contenuto . In tale circostanza è abbastanza facile fregare l'acaro della situazione, consideriamo il codice che potrebbe gestire il parametro :

```
<?php
  $campo = $_GET['campo'];
  $query = "SELECT * FROM tabella_qualsiasi WHERE campo=".$campo;
```

```
?>
```

Considerando che il campo è un numero (detto intero), basta questa piccola modifica al codice x renderlo sicuro :

```
<?php
    $campo = (int)$_GET['campo']; // qui viene eseguito il cast a int
    $query = "SELECT * FROM tabella_qualsiasi WHERE campo=".$campo;
?>
```

Come vedete, il cast a int (`(int)$_GET['campo']`) prende solamente il valore numerico del parametro, escludendo tutto il resto .

La questione si complica nel caso in cui il valore che passiamo in GET (lo stesso discorso vale per il campi il POST) non è numerico, bensì una stringa, in quanto il cast a int ne danneggerebbe i contenuti, sputtanando anche le informazioni che ci servono per effettuare la query .

Esempio :

```
http://www.sito.com/pagina.php?nick=evilsocket
```

e un eventuale injection potrebbe essere :

```
http://www.sito.com/pagina.php?nick=evilsocket' and 1=2 union all select
table_name,1,2 ... N from information_schema.tables where ''='
```

L'ipotetico codice php di gestione sarebbe :

```
<?php
    $nick = $_GET['nick'];
    $query = "SELECT * FROM tabella_qualsiasi WHERE nick='$nick'";
?>
```

Grazie ai padri creatori del php e della rispettiva estensione per i db MySQL, esiste una funzione fatta apposta x questa circostanza, usandola il codice diventerebbe :

```
<?php
    $nick = mysql_real_escape_string( $_GET['nick'] );
    $query = "SELECT * FROM tabella_qualsiasi WHERE nick='$nick'";
?>
```

Dove la funzione `'mysql_real_escape_string'` rimpiazza tutti i caratteri "pericolosi" per una query con un rispettivo "innoquo", rendendo ancora una volta gli attacchi del nostro acaro invani .

Ho lasciato gli xss per ultimi xkè sono i + semplici da evitare ... prendiamo il seguente esempio in considerazione :

```
http://www.sito.com/pagina.php?nick=evilsocket
```

con rispettivo codice :

```
<?php
    $nick = $_GET['nick'];
    echo "Benvenuto tra noi $nick !";
?>
```

Un classico xss potrebbe essere :

```
http://www.sito.com/pagina.php?nick=<script>alert('ciao mondo')</script>
```

In questo caso, basta modificare il codice nel seguente modo :

```
<?php
    $nick = htmlentities($_GET['nick']);
    echo "Benvenuto tra noi $nick !";
?>
```

dove la funzione 'htmlentities' ha lo stesso scopo di 'mysql_real_escape_string', con l'unica differenza che filtra i caratteri html e javascript .

evilsocket