Website Security

Project

Some Techniques What Could Be Danger of Your Website

Paper Draft

CMPT320

Internet and Intranet Security

December, 1st

by

Bence Liska

(2008-12-01)
**Motivation of the Project**

When I was building some Website and I was using some CMS (Content management system - [http://en.wikipedia.org/wiki/Content_management_system](http://en.wikipedia.org/wiki/Content_management_system)) I notified there is a lot of security updates. I was interested why do I have to update my CMS what kind of attacks could be danger of my Websites.

Now, I am getting known a simple few line PHP code could be evadable if I do not watch for the possible attacks onto my Websites.

I would like to present some technique what can be used to hack a website (for example: SQL injection, DOS attacks, etc...) My resource will be covered more popular attacks what is important to develop a website to avoid the possible attacks.

If you know these techniques you can increase the security of your website.
Website Security

First of all we have to define what kind of information we have to protect on our website. Basically every kind of information what the website represents. Here is a general list:

- Customers’ information (credit card numbers, passwords, profiles, etc...)
- Customers’ cookies and sessions
- The administration web forms
- The content of the website
- The database of the website

We cannot imagine what kind of danger things are on the lurk if we don’t want to get to know these dangerous attach techniques what the hackers can use against our website. Let’s look over some interesting techniques what you have to know about them. We are going to speak about these techniques:

- SQL Injection
- Password Cracking
- Cross Site Scripting (XSS)
- Google Hacking

These attacks are only a few of all but if you understand how they work you can see how the “World” is working. You cannot trust anybody, you have to be suspicious.

This Project Paper wants to open your eyes these things and you can develop right secured websites if you know what the preventing methods are. The simple examples are going to show how the technologies used to accept and what the results of them are.

Read on, to learn the basics of how sites and web content management systems are most often hacked, and what you can do to reduce the risk of it happening to you.

If you feel you want to try these examples please be sure do not damage any kind of website of web application.
**SQL Injection**

**Introduction**

This technique involves entering SQL code into web forms, or into browser address field to get access to the website or the database behind the site.

![Log in Address](http://somesite.com/index.asp?id=10 AND id=11)

**The Simple Hack**

In this case if the website do not covered the hacker can get access to the website.

SQL query at the server:

```
"SELECT * FROM users WHERE username = '" + $USRTEXT + "'
AND password = '" + $PASSTEXT + ";"
```

SQL Injection:

```
admin'--
```

The result is:

```
"SELECT * FROM users WHERE username = 'admin' --' AND password = '";
```

Other examples the sample SQL injection:

```
' OR 1=1 --
'

(‘a’=‘a
") OR (‘a”=“a
hi” OR “a”=“a
```
The Injury Hack

In this case the hacker can delete (drop) some tables from the database of the website.

SQL query at the server:
```
"SELECT * FROM DATA WHERE id=" + $ID + ";"
```

SQL Injection:
```
1;DROP TABLE users
```

The result is:
```
"SELECT * FROM DATA WHERE id=1;DROP TABLE users;"
```

Other examples the sample SQL injection:
```
10;DELETE user WHERE id=1;
```

Preventing

A straight-forward, though error-prone way to prevent injections is to escape dangerous characters. For instance, every occurrence of a single quote (') in a parameter must be replaced by two single quotes ("个百分秒") to form a valid SQL string literal. In PHP, for example, it is usual to escape parameters using the function mysql_real_escape_string before sending the SQL query.

Example:
```
$query = sprintf("SELECT * FROM Users where UserName='%s' and Password='%s'",
    mysql_real_escape_string($Username),
    mysql_real_escape_string($Password));
mysql_query($query);
```
Password Cracking

Introduction

Most website encrypts the user’s password (and name) as a hash code. These hashes are in a database table but if somebody have access for this table he/she just find long strings. If the hacker wants to use a user’s account he/she needs to know the user password.

Types

Dictionary attack

Users often choose weak passwords. The directory has a list of single words (given and family names, and any password of 6 characters or less).

“In one survey of MySpace passwords which had been phished, 3.8 percent of passwords were a single word found in a dictionary, and another 12 percent were a word plus a final digit; two-thirds of the time that digit was 1.” (From: Wikipedia)

Brute force attack

This attack tries every possible password to get access to the user account on the website. It needs a lot of time to find the right password especially on a website.

Precomputation (Rainbow table)

Precomputation involves hashing each word in the dictionary (or any search space of candidate passwords) and storing the (plaintext, ciphertext) pairs in a way that enables lookup on the ciphertext field. This way, when a new encrypted password is obtained, password recovery is instantaneous. Precomputation can
be very useful for a dictionary attack if salt is not used properly, and the dramatic decrease in the cost of mass storage has made it practical for fairly large dictionaries.

**Example**

Using [http://www.md5oogle.com](http://www.md5oogle.com) try to find result of a hash.

![md5oogle search result](image)

**Preventing**

The best method of preventing password cracking is to ensure that attackers cannot get access even to the encrypted password. The users can use only strong passwords.
Cross Site Scripting (XSS)

Introduction

Cross Site Scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. Examples of such code include HTML code and client-side scripts. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Vulnerabilities of this kind have been exploited to craft powerful phishing attacks and browser exploits. As of 2007, cross-site scripting carried out on websites were roughly 80% of all documented security vulnerabilities. Often during an attack "everything looks fine" to the end-user who may be subject to unauthorized access, theft of sensitive data, and financial loss.

Types

DOM-based attack

1. Mallory sends the URL of a maliciously constructed web page to Alice, using email or another mechanism.
2. Alice clicks on the link.
3. The malicious web page's JavaScript opens a vulnerable HTML page installed locally on Alice's computer.
4. The vulnerable HTML page contains JavaScript which executes in Alice's computer's local zone.
5. Mallory's malicious script now may run commands with the privileges Alice holds on her own computer.

**Non-Persistent**

1. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password pair and store sensitive information, such as billing information.
3. Mallory crafts a URL to exploit the vulnerability, and sends Alice an email, making it look as if it came from Bob (i.e., the email is spoofed).
4. Alice visits the URL provided by Mallory while logged into Bob's website.
5. The malicious script embedded in the URL executes in Alice's browser, as if it came directly from Bob's server. The script can be used to email Alice's session cookie to Mallory. Mallory can then use the session cookie to steal sensitive information available to Alice (authentication credentials, billing info, etc) without Alice's knowledge.

**Persistent**

1. Bob hosts a web site which allows users to post messages and other content to the site for later viewing by other members.
2. Mallory notices that Bob's website is vulnerable to a type 2 XSS attack.
3. Mallory posts a message, controversial in nature, which may encourage many other users of the site to view it.
4. Upon merely viewing the posted message, site users' session cookies or other credentials could be taken and sent to Mallory's webservers without their knowledge.
5. Later, Mallory logs in as other site users and posts messages on their behalf....
**Preventing**

**Escaping and filtering**

One way to eliminate some XSS vulnerabilities is to encode locally or at the server all user-supplied HTML special characters into character entities, thereby preventing them from being interpreted as HTML. Unfortunately, users of many kinds of web applications (commonly forums and webmail) wish to use some of the features HTML provides. Some web applications such as social networking sites like MySpace and mainstream forum and blog software like WordPress and Movable Type attempt to identify malicious HTML constructs, and neutralize them, either by removing or encoding them. But due to the flexibility and complexity of HTML and related standards, and the continuous addition of new features, it is almost impossible to know for sure if all possible injections are eliminated. Capabilities differ greatly among filtering systems and as of 2007 in Google's case were being written in house. In order to eliminate certain injections, any server-side algorithm must either reject broken HTML, understand how every browser will interpret broken HTML, or (preferably) fix the HTML to be well-formed using techniques akin to those of HTML Tidy.

**Input validation**

Input validation for all potentially malicious data sources is another way to mitigate XSS. This is a common theme in application development (even outside of web development) and is generally very useful. For instance, if a form accepts some field, which is supposed to contain a phone number, a server-side routine could remove all characters other than digits, parentheses, and dashes, such that the result cannot contain a script. Input validation may help to mitigate other injection attacks such as SQL injection as well. While effective for most types of input, there are times when an application, by design, must be able to accept special HTML characters, such as '<' and '>'. In these situations, HTML entity encoding is the only option.

**Cookie security**

Besides content filtering, other methods for XSS mitigation are also commonly used. One example is that of cookie security. Many web applications rely on
session cookies for authentication between individual HTTP requests, and because client-side scripts generally have access to these cookies, simple XSS exploits can steal these cookies. To mitigate this particular threat (though not the XSS problem in general), many web applications tie session cookies to the IP address of the user who originally logged in, and only permit that IP to use that cookie. This is effective in most situations (if an attacker is only after the cookie), but obviously breaks down in situations where an attacker is behind the same NATed IP address or web proxy. IE (since version 6) and Firefox (since version 2.0.0.5) have an HttpOnly flag which allows a web server to set a cookie that is unavailable to client-side scripts but while beneficial, the feature does not prevent cookie theft nor can it prevent attacks within the browser.
Google Hacking

Introduction

Using Google (or other search engines), "search engine hackers" can easily find exploitable targets and sensitive data.

Basic Search Techniques

Since the Google web interface is so easy to use, I won't describe the basic functionality of the http://www.google.com web page. Instead, I'll focus on the various operators available:

- Use the plus sign (+) to force a search for an overly common word. Use the minus sign (-) to exclude a term from a search. No space follows these signs.
- To search for a phrase, supply the phrase surrounded by double quotes (" ").
- A period (.) serves as a single-character wildcard.
- An asterisk (*) represents any word—not the completion of a word, as is traditionally used.

Google advanced operators help refine searches. Advanced operators use a syntax such as the following:

    operator:search_term

Notice that there's no space between the operator, the colon, and the search term.

- The site: operator instructs Google to restrict a search to a specific web site or domain. The web site to search must be supplied after the colon.
• The filetype: operator instructs Google to search only within the text of a particular type of file. The file type to search must be supplied after the colon. Don’t include a period before the file extension.

• The link: operator instructs Google to search within hyperlinks for a search term.

• The cache: operator displays the version of a web page as it appeared when Google crawled the site. The URL of the site must be supplied after the colon.

• The intitle: operator instructs Google to search for a term within the title of a document.

• The inurl: operator instructs Google to search only within the URL (web address) of a document. The search term must follow the colon.

**Examples**

**Example 1:**

Find Wordpress Login Forms:

<table>
<thead>
<tr>
<th>intitle:&quot;WordPress &gt; * &gt; Login form&quot; inurl:&quot;wp-login.php&quot;</th>
<th>intitle:&quot;WordPress &gt; * &gt; Login form&quot; inurl:&quot;wp-login.php&quot;</th>
</tr>
</thead>
</table>

![Search results for Wordpress login forms](image.png)
Example 2:

Find Phpinfo pages:

\[\text{intitle:phpinfo "PHP Version"}\]

Example 3:

Find PhpMyAdmin pages:

\[\text{inurl:main.php Welcome to phpMyAdmin}\]
A Few Defensive Measures

- If you use CMS or other publishing platform or framework you have to update to new versions soon as possible.

- If you have an admin page at your website do not call “admin.php” or “AdminLogin.php” let’s rename it. Call it for example “dog.php” or “hamburger.php”.

- Try to enter some confusing data into your login fields like the sample Injection strings, and any else which you think might confuse the server.

- Escaping and filtering.

- Input validation.

- Cookie security.

- Attackers cannot get access even to the encrypted password.

- The users can use only strong passwords.

- Do a few Google hacks on your name and your website. Just in case...
References

- **How They Hack Your Website: Overview of Common Techniques**  

- **Protecting Your Website Against Hacking**  
  http://www.squidoo.com/protect-your-website-against-hacking

- **Hacking Techniques**  
  http://infinityexists.com/

- **Denial-of-service attack (Wikipedia article)**  
  http://en.wikipedia.org/wiki/Dos_(attack)

- **Trends in Denial of Service Attack Technology**  

- **Denial of Service Attacks**  
  http://www.cert.org/tech_tips/denial_of_service.html

- **SQL Injection (Wikipedia article)**  

- **(more) Advanced SQL Injection**  

- **SQL Injection Attacks by Example**  
  http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/

- **SQL Injection Walkthrough**  
  http://www.securiteam.com/securityreviews/5DP0N1P76E.html

- **Episode 11 - Website Hacking - SQL Injection**  
- **SQL Injection Cheat Sheet**

- **Password Cracking (Wikipedia article)**

- **Password Cracking with Rainbowcrack and Rainbow Tables**

- **Episode 10 - Hacking Basics - MD5**

- **Cross-site Scripting (Wikipedia article)**

- **The Cross Site Scripting (XSS) FAQ**

- **Cross Site Scripting Understanding the vulnerability and payload**

- **Cross Site Scripting Attack**
  [http://www.acunetix.com/websitesecurity/cross-site-scripting.htm](http://www.acunetix.com/websitesecurity/cross-site-scripting.htm)

- **Episode 13 - Website Hacking - XSS**

- **Google hacking (Wikipedia)**

- **Google Hacking Database (GHDB)**

- **Advanced Operators**

- **Google Hacking - Is your web application secure?**
- **Google Hacking Mini-Guide**

- **Google hacking**