# Hacking Techniques: Web Application Security

**Shynlie Simmons**

**East Carolina University**
**November 27, 2005**

**RUNNING HEAD: Hacking Techniques**

## Abstract

This paper focuses on hacking techniques of web applications and how the implementation of security through programming can keep intruders from wreaking havoc on your system. The paper will define a web application and discuss the architecture of the web application, as it will explain the multiple tier theory. The paper will discuss security in web applications and will look at basic rules in information security planning. The paper will look at seven steps in web application hacking and the top ten vulnerabilities that criminals can exploit in order to gain access and take control of a computer system. It is hoped that security professionals will take a close look at this seriously dangerous security risk in order to help close the security holes that could and do exist in web applications.

## Introduction

Since 1984 and the commercialization of the Internet, the number of computers linked to the Internet as hosts has grown from approximately 1000 in 1984 to over 150 million[1] in 2005. This growth means that there are more than 150 million people providing, using and sharing resources and information via the public network. These users and hosts are located all over the globe and are in every country that has access to the Internet. Different types of information and resources are available in many different forms, from informational web sites (static content), to interactive sites that display different results depending on the interaction (dynamic content) of the user. As with every population, you will have "good guys" and "bad guys" and every web site can fall victim to criminal activities. This paper will explain some of the different ways that criminals can do harm or get unauthorized information and how programmers can stop this by implementing security into their programming code.

## What is a Web application

When web pages were first implemented they only displayed unchanging information, which is called static content. They may have included hyperlinks that would point to other web sites or web pages, which made it easy for users to get information from multiple or related web sites. These web sites didn't offer any interactivity with users and didn't require user input. Techniques and programming languages were developed that allows newer web pages to include text boxes, check boxes, radio buttons, drop down lists and command buttons so that the user can provide data, information and commands to the web site. These web sites that change consists of dynamic web content and are also called interactive. "Dynamic content is material on a Web page that is added or altered, usually after the page has been loaded by the Web browser and usually in response to actions or requests by the user".[2]

## *Basic Architecture*

The basic architecture of a web application can be thought of in terms of structure and language. The structure for interactive applications is usually multi-leveled and can be written in multiple languages.

### Multi-tier

Web application architecture can be thought of as a "multi" or "nTier" structure. This means there are several layers needed to build a web application and usually made up of at least three layers; the presentation tier (front end), application (business logic) and data tiers (back end).[3]

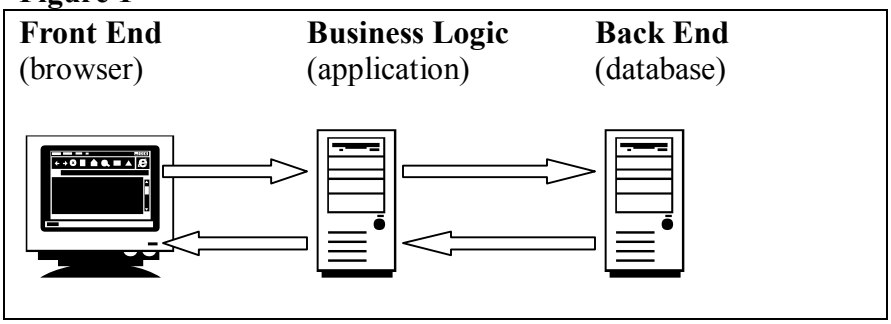
The presentation tier, which is very commonly referred to the "front end", is portion of the application that the user interacts with directly and is normally displayed by a software know as the "browser". This user interaction portion is referred to as the user interface.

The application or business logic tier of the web application is the main part of the program where the programming code is written and stored. This is the brain of the application where all the commands and instructions are carried out.

The data tier is the portion of the web application that stores all of the data and is commonly referred to as the "back end". Most applications store the data in a type of file management system called a database, which has made the interaction between user and data possible.

Figure 1 below, illustrates the flow as the user loads their browser into memory, then requests a web page from the application server, the application server then will request information from the back end or database and the data is sent back through the application and rendered in the browser by the HTML code.

**Figure 1**




## Languages

Web applications can be written in many languages both high-level and scripting.

Hypertext Markup Language (HTML) is a markup language that is typed into a text document and is used with an interpreter (i.e. Netscape Navigator, Microsoft Internet Explorer, or Apple Safari, etc.) to create static content and to display the results of a dynamic content web page. Languages such as ASP.NET and J2EE are high-level languages used to write large-scale web applications that are more complex and harder to code. These languages are used for heavy duty computing. Procedural languages such as ASP, Cold Fusion, PHP and scripting languages such as JavaScript and VBScript used for lighter, less complex web page content and are the most common languages used.[4]

## Security

Since the Internet has more criminals lurking around trying to steal important information, there have to be laws, standards and practices put in place to help protect the confidentiality, integrity and availability of data that is accessed through the Internet.

"When consumers open an account, register to receive information or purchase a product from your business, it's very likely that they entrust their personal information to you as part of the process. If their information is compromised, the consequences can be

far – reaching: consumers can be at risk of identity theft, or they can become less willing – or even unwilling – to continue to do business with you."[5]

The Federal Trade Commission has suggested that you follow some basic rules in information security planning:[5]

- Identify internal and external risks to the security
- Design and implement safeguards to control the risks
- Periodically monitor and test the safeguards
- Adjust the security plan according to the results of testing
- Oversee the information handling practices of service providers and business partners

As part of their plan for determining and identifying security risks, they suggest that you look at the "*The 10 Most Critical Web Application Security Vulnerabilities"*.[6] This list is useful in tapering your web application to best practices. This can be done either in the design or maintenance phase of the System Development Life Cycle.

According to the OSWAP, "organizations need to establish information security policy informed by relevant national legislation, industry regulation, merchant agreements, and subsidiary best practice guides, such as OWASP"[4]. A guide for doing this in the United States can be illustrated in the Table 1.

**Table 1**

| Level | Compare To | Responsibility of |
|---|---|---|
| National Legislation | Sarbanes Oxley | Legal Begal |
| Industry Regulations | UCCC<br>Visa PCI | Legal Beagle |
| National Standards | NIST – SP800 | Security Reviewer |
| Industry Standards | OWASP Criteria<br>ITIL<br>COBIT<br>OWASP Guide<br>OWASP Testing | Business |
| Organization | Information Security Policy | CSO, CIO |

Security in the past has been thought of a network issue, but in fact some of the most dangerous vulnerabilities are in the applications themselves.

## What is vulnerable?

A security hole is defined as “the program [that] has a flaw that allows an attacker to "exploit it." Thus comes the "exploit" that denotes a program, or technique to take advantage of the flaw, or "vulnerability."[7] In the past, security was considered a networking topic and threat countermeasures were implemented on the network and operating system. Those measures included firewalls and intrusion detection systems. However “traditional network-based countermeasures, such as firewalls and IDS, do very little in the way of offering protection to web-based applications.”[8]

The problem is made worse by unsecured application development as most experienced developers are not knowledgeable about the new web application environments, but are immerging from the legacy hard coded procedural applications and have not yet caught on to the new developing script programs. Another challenge met by application development is the “time to market” mind frame of marketing, this is the idea that the product must be on the market as soon as possible, meaning that testing is not done to alleviate all bugs in the system prior to market. “Hackers are sometimes able to anticipate inadequacies and the coding practices adopted by programmers. Often, the ``speed to market" attitude pushes application developers to overlook standard and secure coding practices.” [8]

Lack of knowledge by new or inexperienced programmers, coupled with quick and shoddy programming, makes web applications an easy target for criminals.

### *The Business Logic Tier*

"The root cause of most of today's application vulnerabilities is their heritage: most Web applications were developed and tested using methodologies created for client-server applications. In some cases, they started out as client-server applications and were 'ported' to the Web, a process that often consisted of little more than reproducing the client GUI in a Web browser. In other cases, they were developed for the Web but their coding and testing practices did not incorporate the unique challenges of Web security." [9]

This code is harder to see because it resides on the server and must be reverse engineered to be able to see the code. In order to "hijack" or take over the session, you would have to have network access first, and then have to be able to get into the code. However with the conversion from legacy server-side applications running on the business logic unit, to client-side applications, there could be holes that criminals can gain access to and manipulate for their deceitful gain.

### *The Presentation Tier*

Heavy server based applications were hard to break into because they needed reverse engineering in order to see the code and mostly you had to have a copy of the software to do this, so most users had to be privy to that in order to get a copy. However, with browsers this is not true, as they are very easy to gain access to and manipulate, as the source code is made available to anyone who knows how to right click. Also the use of cookies can act as a way the criminal can steal private information from the browser.

### *The Data Tier*

"Thinking through the layers of security in our environment, I realized that the weakest link in the chain is at the application layer, which is where I see database security fitting in. Much

attention has been given to auditing firewall rules, turning off unneeded services on servers and patching operating systems, internetwork operating systems and various applications, such as Internet Explorer. But it seems that not much attention has been given to database security and auditing."[10]

The database is the place where all of the organization's data and information is stored. Using SQL injection, a criminal can copy, delete, change or retrieve information stored in the database. This is one of the most dangerous areas that the system could be vulnerable.

## Hacking Techniques

There are multiple ways a criminal can target the web application tiers. According to "*Hacking Techniques in Wired Networks*"[11] an attack will go through several steps and these actions will use more than one "hacking" technique. Seven of these are outlined below

### *Reconnaissance*

In this step the criminal will gather information about the target system. In web applications this can include the usernames, passwords, input parameters, programming or script languages, type of server and operating system.

### *Probe*

In this phase the criminal detects weaknesses in the system. SQL injection is the most used here for web applications in order to find out what permissions are placed on the system and where the security holes exist.

### *Toehold*

Once the vulnerabilities are determined in the probe stage, then the criminal will proceed to the toehold phase where the intruder begins to enter the system, build a connection, usually referred to a session, search the system for information and begin exploiting the vulnerability

### *Advancement*

This is the phase in which the criminal will move look for configuration errors and move from an unprivileged account into a privileged, or move from a normal user with little access to one of an administrator or super user, in which case the criminal would have full access to create, delete, modify, retrieve or move files and information.

### *Stealth*

The stealth stage is a step that is taken to mask the intruder's presence in the system. This is done by accessing and modifying local log files to remove any evidence that would alert or suggest an attack has taken place.

### *Listening Post*

In the listening post phase, the intruder will set up a "backdoor", which is a malicious program that will ensure that future activities will not be logged. These programs are called stealth or backdoor tools, or sniffer. These will report false information on the file access and processes on the network and system in order to disguise the intruder's subsequent entries.

### *Takeover*

The takeover stage is used to expand the control from one system to others on the same or connecting networks. The intruder can use a sniffer to detect other information on other hosts such as usernames and passwords. This can be used to find host machines that trust the computer the criminal is currently attacking so that the intruder can get access to the other hosts.

These steps can be done in a number of ways and use a number of tools for each. Some of these are enabled simply by poor programming and should be controlled by the programmer at the design phase.

## Top Ten Vulnerabilities

The Open Web Application Security Project (OWASP) is a non-profit open group that is focused on understanding and improving web applications security. This group has published a

document entitled *Ten Most Critical Web Application Security Vulnerabilities,* which states that most network security ignores the content of HTTP traffic and therefore creates no policing of traffic through port 80. This means that there are many vulnerabilities that exist. The top ten are outlined in the document are listed below:[6]

1. Unvalidated Parameters
2. Broken Access Control
3. Broken Account and Session Management
4. Cross-Site Scripting Flaws
5. Buffer Overflows
6. Command Injection Flaws
7. Error Handling Problems
8. Insecure Use of Cryptography
9. Remote Administration Flaws
10. Web and Application Server Misconfiguration

## Which vulnerabilities can be controlled by programmers

Now we know where vulnerabilities exist and the steps criminals take to access, steal or deface information, we can look at some of the measures web site developers can take in implementing security to protect an organization's assets.

### *Data Validation*

The most common weakness in web applications is the failure to properly validate input from the user (client or environment) in URL, query strings, form fields, hidden fields, cookies and headers. User input should be checked for items such as strongly typed data, correct syntax, data within length boundaries, data that contains only permitted characters, or if numeric data is within range boundaries. Web applications use input from users in order to determine how to react to an HTTP request, if the data does not meet these criteria, then the application my be subject to SQL injection or buffer overflow.

## SQL Injection

"SQL injection is a type of security exploit in which the attacker adds SQL statements through a web application's input fields or hidden parameters to gain access to resources or make changes to data."[12] A resource field is usually considered a column in a database that holds important data about an individual, customer or entity. Validation is key to making sure these SQL statements don't contain criminal code necessary to illegally obtain unauthorized information. This can be done using any database, as no any one vendor is exempt. "SQL injection is not a defect of Microsoft SQL Server – it is also a problem for every other database vendor as well."[13]

Let's take a look at how SQL injection works by looking at a simple database with user names and passwords. In a normal database, the table named "Login" would look like Table 2:

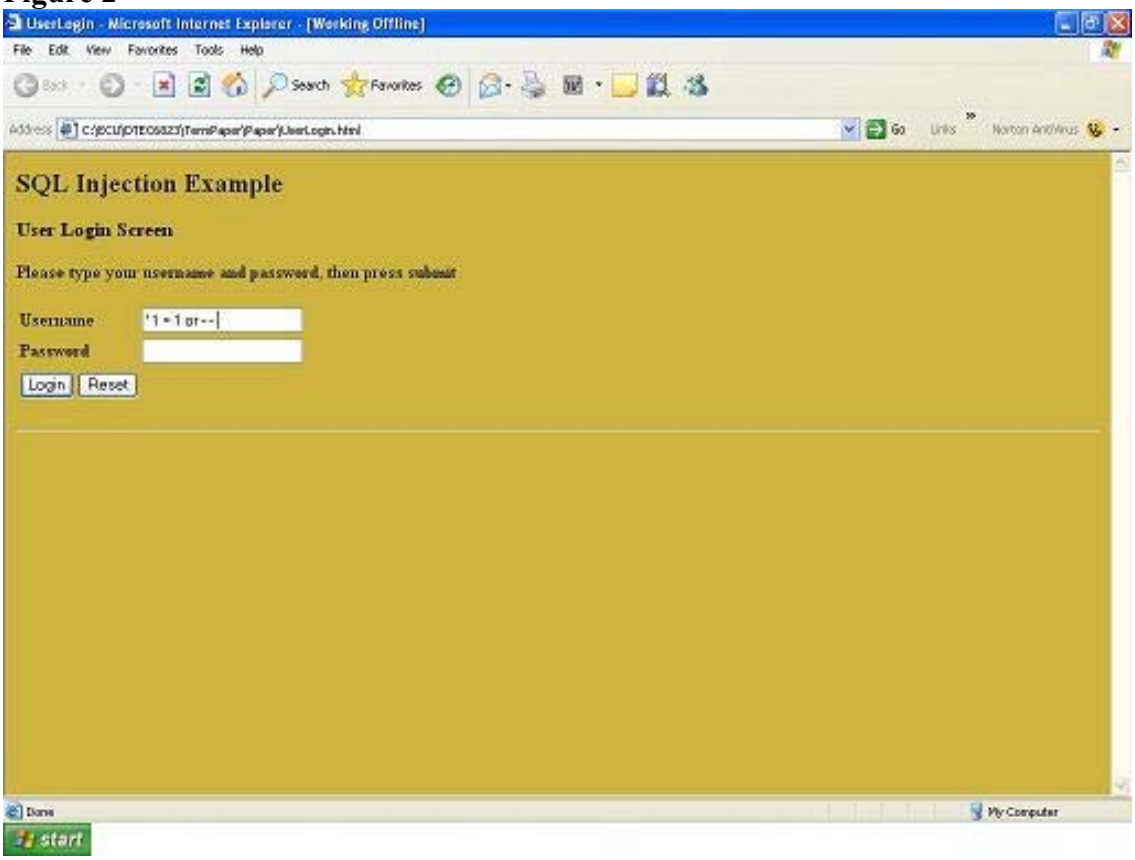
**Table 2**

| PrimaryID | Username | Password |
|---|---|---|
| 1 | mickey123 | mic123 |
| 2 | minnie456 | min456 |
| 3 | donald789 | don789 |
| 4 | goofy012 | goo012 |

The primary ID is a unique identifier for the row and is only used by the database and programmers to identify that record. The Username field contains each user's identity and the Password field contains the user's chosen password.

Now let's look at a user sign on screen shown in Figure 2.

**Figure 2**




If the form field of the web page for the username didn't validate and allowed us to type " ' 1 = 1 or - - " then the resulting query would be read as: [12]

```
SELECT * FROM Login WHERE Username = ' ' or 1=1- - AND
password = '  '
```

In order to understand this, you need a little knowledge of queries. Since understanding of queries is beyond the scope of this paper, we will only explain what this sample query means as shown in Table 3

**Table 3**

| Statement or Code | Description |
|---|---|
| **SELECT** | Instructs the database engine to retrieve records. |
| * (wildcard character) | Instructs the database engine to select **all** fields |
| **FROM Login** | Instructs the database engine to retrieve records from the table named Login |
| **WHERE** | Provides a criteria to match when selecting records |
| **“ “** | Evaluated by the database engine will cause all fields that are null ‘ ‘ or empty to be returned (which are really useless here, except that it negates the ‘ in the SQL statement that would be created in a normal query) |
| **or** | Instructs the code to look for another criteria in addition to the first criteria. |
| **1=1** | Evaluated by the database engine to be true which will then yield all records in the table. |
| **- -** | Acts as a comment in code and is not executed. |

At this point a query has been sent to the database engine, which will yield all records in the database. Using other code in addition with this, will yield a list of all usernames and passwords in the database. Once this data is collected, you can gain connectivity to the database and have completed the toehold phase of hacking

Once the toehold phase is complete, the criminal can move into the advancement phase by writing all of the username and passwords into a file and the file can be transferred to the criminal’s computer. Other phases of hacking can be completed using SQL injection and database query commands meaning that multiple servers can be accessed and compromised. Table 4 shows an example of query commands and their functions, giving an indication of the power of this type of exploit.

**Table 4**

| Command | Function |
|---|---|
| SELECT | Selects rows (records) |
| INSERT INTO | Inserts rows (records) |
| DELETE | Deletes rows (records) |
| UPDATE | Changes rows (records) |
| CREATE TABLE | Creates a new table |

The database is not the only entity that can be affected here, the criminal can also access the network as well as the operating system at this point. An example of starting an FTP service using SQL injection is as follows:[12]

```
'; exec master..xp_servicecontrol 'start','FTP
Publishing' –
```

SQL statements can be written to bypass the signatures of intrusion detection systems; however the discussion of this goes beyond the scope of this paper. The best defense is to check all input and validate at all tiers.

## Buffer Overflow

A buffer overflow can be described as sending too much information to a memory location used to hold variable values. It is important to make sure the data typed into the input form field is within a certain range for length or size to avoid an overflow. "Attackers use buffer overflows to corrupt the execution stack of a web application. By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code - effectively taking over the machine." [4]

One example of this is the IFRAME flaw, where "the attackers took over the load balancer for some advertising sites that posted banner ads on a variety of other websites. If you viewed any of these ads at any of these sites with a vulnerable browser, you'd get the Bofra worm installed on

your machine."[14] This would be a very ugly scenario for your customers not to mention the end of the year financials.

### Cross-Site Scripting Flaws

Cross-site scripting flaws can have damaging effects similar to the IFRAME flaw discussed in the buffer overflow section earlier. In cross-site scripting, the criminal enters code in any form such as HTML, JavaScript, VBScript or Flash. This code is then stored in the field of the database. Once another user retrieves this data from this field, if the code is malicious it could transmit a virus or worm into the users computer or system. "A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user."[15]

This can be prevented by making sure the data that is retrieved and displayed in the browser doesn't contain any special characters such as ", &, <, >, into, amp, &lt, and &gt. These characters used in conjunction with scripting code can be dangerous to applications users and should not be allowed.

### Command Injection

Command injection is similar to SQL injection, except you enter a command into the input field rather than simple code. These commands can be codes such as: exec(), passthru(), system(), popen() and the backtick operator (`). When included, these commands can execute arbitrary commands and wreak havoc on any system.

## *Error Handling Problems*

Some of the most useful information a programmer can provide to a criminal is in the form of an error message. These messages can tell attackers information that will help them to determine the operating system, database and programming language used in each tier. Criminals are

usually well versed in various operating system and programming languages so that they can gain control of any system. Once they know what language is used then they will know what type of injection they can place into an input field to gain entry and control.

Table 5 is an example of different syntax used in two different databases for the same action.

**Table 5**

| | MS SQL (T-SQL) | MS Access |
|---|---|---|
| To concatenate a string | + | & |
| To test for a null value | IsNull( ) | Iff(IsNull( ) ) |
| To find a position in a string variable | CHARINDEX | InStr( ) |

Error messages should be vague and not conceal any information about the type of application used for blocking attacks, what operating system is employed or programming languages used to create the applications. It is suggested that an error ID number should be generated for the user to reference if they need to call a help desk in response to a valid error.

## *Cookie Poisoning*

A cookie is a computer file that contains data that is written to the client computer's hard drive or cache memory when accessing a web application through a browser. This file contains information such as username, password, token and session data. Criminals can use this information to "hijack" a web session and use the client computer for malicous deeds.

There are two types of cookies, persistant and non-persistant. They are very similar in function but have two major differences. The persistent cookie has an expiration date and is stored on the client hard drive until the expiration date, at which time it is deleted. A non-persistant cookie is a data file stored in the client computer's memory and is erased when the web browser is removed from memory (or closed by the user).

The most secure type of cookie is the non-persistant cookie, as it only exists for a short amount of time and so has less chance of being accessed by a criminal. The security measures you can take when using cookies are:[8]

- Use non-persistent cookies instead of persistent cookies.
- If you must use persistent cookies, then specify a short duration for the cookie's life. The longer the time until cookie expiration, the larger the risk.
- Avoid application features that use persistent cookies to store privacy-related information. Example: ``Please check to remember user name and password."
- Use the secure tag, so that the cookie is sent only if a secure channel (https) is being used.
- Encrypt the information in the cookies. Some web sites split one cookie into many cookies that are further encrypted.

## Conclusion

There are many different ways a criminal can break into a system and wreak havoc on a network or computer system. It is up to the web application coder to do their part in making sure the applications they design are not vulnerable to any know threats. The World Wide Web Consortium has published some basic coding rules to follow in designed a web application and are outlined below. The expanded version can be found at www.w3j.com/7/s3.garfinkel.wrap.html.[16]

1) Carefully design the program before you start.
2) Check all values provided by the user.
3) Check arguments that you pass to operating system functions.
4) Check all return codes from system calls.
5) Have internal consistency-checking code.
6) Include lots of logging.
7) Make the critical portion of your program as small and as simple as possible.
8) Read through your code.
9) Always use full pathnames for any filename argument, for both commands and data files.
10) Rather than depending on the current directory, set it yourself.
11) Test your program thoroughly.
12) Be aware of race conditions. These can be manifest as a deadlock or as failure of two calls to execute in close sequence.
13) Don't have your program dump core except during your testing.

14) Do not create files in world-writable directories. If your CGI script needs to run as the *nobody* user, then have the directory in which it needs to create files owned by the *nobody* user. (This also applies to the */tmp* directory.)
15) Don't place undue reliance on the source IP address in the packets of connections you receive. Such items may be forged or altered.
16) Include some form of load shedding or load limiting in your server to handle cases of excessive load.
17) Put reasonable time-outs on the real time used by your CGI script while it is running.
18) Put reasonable limits on the CPU time used by your CGI script while it is running.
19) Do not require the user to send a reusable password in plaintext over the network connection to authenticate himself or herself.
20) Have your code reviewed by another competent programmer (or two, or more).
21) "Whenever possible, steal code."

## References

1. Shelly, G.B., Cashman, T.J., and Vermaat, M.E. (2004). *Discovering Computers 2005, A Gateway to Information*. Boston, MA: Course Technology.
2. Carey, P. (2001). *Creating Web Pages with HTML and Dynamic HTML*. Boston, MA: Course Technology
3. Stasiak, K. (2002). Web Application Security. *Information Systems Control Journal*, vol. 6. Retrieved Nov 27 2005 from http://www.isaca.org/Content/ContentGroups/Journal1/20023/Web_Application_Security.htm
4. The Open Web Application Security Project. (2005). *A Guide to Building Secure Web Applications and Web Services* (2.0 Black Hat ed.) van der Stock, Andrew: Contact. Retreived Nov 27 2005 from http://www.owasp.org/documentation/guide.html
5. Federal Trade Commission. (2003). *Security Check: Reducing Risks to your Computer Systems*. Washington, DC: Retrieved Nov 27 2005 from http://www.ftc.gov/bcp/conline/pubs/buspubs/security.htm
6. The OWASP Foundation. (2004). *OWASP Top Ten Most Critical Web Application Security Vulnerabilities*. Williams, J.: Author. Retreived Nov 27 2005 from http://www.owasp.org/documentation/topten.html.
7. Al-Herbish, T., & Roozemaal, P. (1999). Secure *UNIX Programming FAQ*. Whitefang Dawt Kawm. Retrieved Nov 27 2005 from http://www.whitefang.com/sup/secure-faq.html
8. Gaur, N. (2000, April 1). Assessing the Security of Your Web Applications. *Linux Journal*. Retrieved Nov 27 2005 from http://www.linuxjournal.com/article/3855
9. Stern, A. (2004). Web Application Vulnerabilities. *The ISSA Journal*. Retrieved on Nov 27 2005 from http://www.isaca.org/Content/ContentGroups/Journal1/20023/Web_Application_Security.htm
10. Kelly, C. (2005, July 25). Getting Started on Database Security. *Security Manager's Journal*. Computerworld Inc. Retrieved Nov 27 2005 from http://www.computerworld.com/securitytopics/security/story/0,10801,103448,00.html?source=x490
11. Gu, Q., Liu, P. and Chu, C. H. (2004) "Hacking Techniques in Wired Networks" in Hossein Bidgoli et al. (eds.), *Handbook of Information Security*, John Wiley & Sons, New York.
12. The OWASP Foundation. (2005). *Advanced SQL Injection*. Chapela, V.: Author. Retrieved Nov 27 2005 from www.owasp.org/docroot/owasp/misc/Advanced_SQL_Injection.ppt
13. Cerrudo, C. (2005). Manipulating Microsoft SQL Server Using SQL Injection. *Microwave Journal Research Center*. Application Security Inc. Aug 1, 2005. Retrieved Nov 27 2005 from http://research.mwjournal.com/detail/RES/1124462486_292.html
14. Skoudis, E. & Fisher, M. (2005). Cutting *Edge Hacker Techniques*. Sans Institute. Web Cast dated May 4 2005. Retrieved Nov 27 2005 from http://www.sans.org/webcasts/show.php?webcastid=90530

15. Check Point Software Technologies Ltd. (2004). A *Practical Guide to Web Application Security:* Mitigating the OWASP Ten Most Critical Web Application Security Problems With Check Point Solutions. [White paper] Retrieved Nov 27, 2005 from http://www.itpapers.com/whitepaper.aspx?compid=3349&dtid=1&docid=127836

16. Garfenkel, S. & Spafford, G. (1997). Secure *AGI/CGI Programming*. World Wide Web Journal. vol 2, issue 3. Retrieved Nov 27 2005 from http://www.w3j.com/7/s3.garfinkel.wrap.html