



VulnoCity Website Audit
www.onlinehackscan.com

18 August, 2008

Detailed Scan Report

Scan of http://testasp.scandemo.com/

Scan details

Scan information	
Starttime	7/25/2008 5:39:47 PM
Finish time	7/25/2008 5:41:46 PM
Scan time	1 minutes, 59 seconds
Profile	default

Server information	
Responsive	True
Server banner	Microsoft-IIS/6.0
Server OS	Windows
Server technologies	ASP.NET

Threat level



Vulnocity HackScan Threat Level 3

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Alerts distribution

Total alerts found	200
High	58
Medium	1
Low	99
Informational	42

Knowledge base

List of files with inputs

- **/templatize.asp** - 1 inputs
- **/search.asp** - 1 inputs
- **/login.asp** - 3 inputs
- **/register.asp** - 5 inputs
- **/showforum.asp** - 1 inputs
- **/showthread.asp** - 1 inputs

Alerts summary

Blind SQL/XPath injection	
Affects	Variations
/showforum.asp	1
/showthread.asp	1

Cross Site Scripting	
Affects	Variations
/search.asp	23

Script source code disclosure

Affects	Variations
/templatize.asp	3

SQL injection

Affects	Variations
/login.asp	6
/register.asp	12
/search.asp	2
/showforum.asp	5
/showthread.asp	5

Cookie manipulation

Affects	Variations
/search.asp	1

Application error message

Affects	Variations
/login.asp	2
/register.asp	36
/search.asp	1
/showforum.asp	10
/showthread.asp	10
/templatize.asp	12

Possible sensitive directories

Affects	Variations
/cgi-bin	1
/html	1

User credentials are sent in clear text

Affects	Variations
/login.asp	13
/register.asp	13

GHDB: Typical login page

Affects	Variations
/login.asp	16

Password type input with autocomplete enabled

Affects	Variations
/login.asp	13
/register.asp	13

Alert details

🚨 Blind SQL/XPath injection

Severity	High
Type	Validation
Reported by module	MultiRequest parameter manipulation

Description

This script is possibly vulnerable to SQL/XPath Injection attacks.

SQL injection is a vulnerability that allows an attacker to alter backend SQL statements by manipulating the user input. An SQL injection occurs when web applications accept user input that is directly placed into a SQL statement and doesn't properly filter out dangerous characters.

This is one of the most common application layer attacks currently being used on the Internet. Despite the fact that it is relatively easy to protect against, there is a large number of web applications vulnerable.

XPath Injection is an attack technique used to exploit web sites that construct XPath queries from user-supplied input.

Impact

An unauthenticated attacker may execute arbitrary SQL/XPath statements on the vulnerable system. This may compromise the integrity of your database and/or expose sensitive information.

Recommendation

Your script should filter metacharacters from user input.
Check detailed information for more information about fixing this vulnerability.

Detailed information

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

SQL injection mitigations

We believe that web application developers often simply do not think about "surprise inputs", but security people do (including the bad guys), so there are three broad approaches that can be applied here.

Sanitize the input

It's absolutely vital to sanitize user inputs to insure that they do not contain dangerous codes, whether to the SQL server or to HTML itself. One's first idea is to strip out "bad stuff", such as quotes or semicolons or escapes, but this is a misguided attempt. Though it's easy to point out some dangerous characters, it's harder to point to all of them.

The language of the web is full of special characters and strange markup (including alternate ways of representing the same characters), and efforts to authoritatively identify all "bad stuff" are unlikely to be successful.

Instead, rather than "remove known bad data", it's better to "remove everything but known good data": this distinction is crucial. Since - in our example - an email address can contain only these characters:

```
abcdefghijklmnopqrstuvwxy  
z  
ABCDEFGHIJKLMN  
OPQRSTUVWXYZ  
0123456789  
@. - _ +
```

There is really no benefit in allowing characters that could not be valid, and rejecting them early - presumably with an error message - not only helps forestall SQL Injection, but also catches mere typos early rather than stores them into the database.

Be aware that "sanitizing the input" doesn't mean merely "remove the quotes", because even "regular" characters can be

troublesome. In an example where an integer ID value is being compared against the user input (say, a numeric PIN):

```
SELECT fieldlist
  FROM table
 WHERE id = 23 OR 1=1;  -- Boom! Always matches!
```

In practice, however, this approach is highly limited because there are so few fields for which it's possible to outright exclude many of the dangerous characters. For "dates" or "email addresses" or "integers" it may have merit, but for any kind of real application, one simply cannot avoid the other mitigations.

Escape/Quotesafe the input

Even if one might be able to sanitize a phone number or email address, one cannot take this approach with a "name" field lest one wishes to exclude the likes of Bill O'Reilly from one's application: a quote is simply a valid character for this field.

One includes an actual single quote in an SQL string by putting two of them together, so this suggests the obvious - but wrong! - technique of preprocessing every string to replicate the single quotes:

```
SELECT fieldlist
  FROM customers
 WHERE name = 'Bill O''Reilly';  -- works OK
```

However, this naive approach can be beaten because most databases support other string escape mechanisms. MySQL, for instance, also permits `\` to escape a quote, so after input of `\'; DROP TABLE users; --` is "protected" by doubling the quotes, we get:

```
SELECT fieldlist
  FROM customers
 WHERE name = '\\'; DROP TABLE users; --';  -- Boom!
```

The expression `\"` is a complete string (containing just one single quote), and the usual SQL shenanigans follow. It doesn't stop with backslashes either: there is Unicode, other encodings, and parsing oddities all hiding in the weeds to trip up the application designer.

Getting quotes right is notoriously difficult, which is why many database interface languages provide a function that does it for you. When the same internal code is used for "string quoting" and "string parsing", it's much more likely that the process will be done properly and safely.

Some examples are the MySQL function `mysql_real_escape_string()` and perl DBD method `$dbh->quote($value)`. These methods must be used.

Use bound parameters (the PREPARE statement)

Though quotesafing is a good mechanism, we're still in the area of "considering user input as SQL", and a much better approach exists: bound parameters, which are supported by essentially all database programming interfaces. In this technique, an SQL statement string is created with placeholders - a question mark for each parameter - and it's compiled ("prepared", in SQL parlance) into an internal form. Later, this prepared query is "executed" with a list of parameters:

Example in perl

```
$sth->execute($email);
```

Thanks to Stefan Wagner, this demonstrates bound parameters in Java:

Insecure version

```
ResultSet rs = s.executeQuery("SELECT email FROM member WHERE name = "
                               + formField); // *boom*
```

Secure version

```
"SELECT email FROM member WHERE name = ?");  
ps.setString(1, formField);  
ResultSet rs = ps.executeQuery();
```

Here, \$email is the data obtained from the user's form, and it is passed as positional parameter #1 (the first question mark), and at no point do the contents of this variable have anything to do with SQL statement parsing. Quotes, semicolons, backslashes, SQL comment notation - none of this has any impact, because it's "just data". There simply is nothing to subvert, so the application is largely immune to SQL injection attacks.

There also may be some performance benefits if this prepared query is reused multiple times (it only has to be parsed once), but this is minor compared to the enormous security benefits. This is probably the single most important step one can take to secure a web application.

Limit database permissions and segregate users

In the case at hand, we observed just two interactions that are made not in the context of a logged-in user: "log in" and "send me password". The web application ought to use a database connection with the most limited rights possible: query-only access to the members table, and no access to any other table.

The effect here is that even a "successful" SQL injection attack is going to have much more limited success. Here, we'd not have been able to do the UPDATE request that ultimately granted us access, so we'd have had to resort to other avenues.

Once the web application determined that a set of valid credentials had been passed via the login form, it would then switch that session to a database connection with more rights.

It should go almost without saying that sa rights should never be used for any web-based application.

Use stored procedures for database access

When the database server supports them, use stored procedures for performing access on the application's behalf, which can eliminate SQL entirely (assuming the stored procedures themselves are written properly).

By encapsulating the rules for a certain action - query, update, delete, etc. - into a single procedure, it can be tested and documented on a standalone basis and business rules enforced (for instance, the "add new order" procedure might reject that order if the customer were over his credit limit).

For simple queries this might be only a minor benefit, but as the operations become more complicated (or are used in more than one place), having a single definition for the operation means it's going to be more robust and easier to maintain.

Note: it's always possible to write a stored procedure that itself constructs a query dynamically: this provides no protection against SQL Injection - it's only proper binding with prepare/execute or direct SQL statements with bound variables that provide this protection.

Isolate the webserver

Even having taken all these mitigation steps, it's nevertheless still possible to miss something and leave the server open to compromise. One ought to design the network infrastructure to assume that the bad guy will have full administrator access to the machine, and then attempt to limit how that can be leveraged to compromise other things.

For instance, putting the machine in a DMZ with extremely limited pinholes "inside" the network means that even getting complete control of the webserver doesn't automatically grant full access to everything else. This won't stop everything, of course, but it makes it a lot harder.

Configure error reporting

The default error reporting for some frameworks includes developer debugging information, and this cannot be shown to outside users. Imagine how much easier a time it makes for an attacker if the full query is shown, pointing to the syntax

error involved.

This information is useful to developers, but it should be restricted - if possible - to just internal users.

Affected items

/showforum.asp

Details

The GET variable **id** is vulnerable.

/showthread.asp

Details

The GET variable **id** is vulnerable.

Cross Site Scripting

Severity	High
Type	Validation
Reported by module	Parameter manipulation

Description

This script is possibly vulnerable to Cross Site Scripting (XSS) attacks.

Cross site scripting (also referred to as XSS) is a vulnerability that allows an attacker to send malicious code (usually in the form of Javascript) to another user. Because a browser cannot know if the script should be trusted or not, it will execute the script in the user context allowing the attacker to access any cookies or session tokens retained by the browser.

Impact

Malicious users may inject JavaScript, VBScript, ActiveX, HTML or Flash into a vulnerable application to fool a user in order to gather data from them. An attacker can steal the session cookie and take over the account, impersonating the user. It is also possible to modify the content of the page presented to the user.

Recommendation

Your script should filter metacharacters from user input.

Detailed information

Quote from The Cross Site Scripting FAQ - <http://www.cgisecurity.com/articles/xss-faq.shtml>

Introduction

Websites today are more complex than ever, containing a lot of dynamic content making the experience for the user more enjoyable. Dynamic content is achieved through the use of web applications which can deliver different output to a user depending on their settings and needs. Dynamic websites suffer from a threat that static websites don't, called "Cross Site Scripting" (or XSS dubbed by other security professionals). Currently small informational tidbits about Cross Site Scripting holes exist but none really explain them to an average person or administrator. This FAQ was written to provide a better understanding of this emerging threat, and to give guidance on detection and prevention.

"What is Cross Site Scripting?"

Cross site scripting (also known as XSS) occurs when a web application gathers malicious data from a user. The data is usually gathered in the form of a hyperlink which contains malicious content within it. The user will most likely click on this link from another website, instant message, or simply just reading a web board or email message. Usually the attacker will encode the malicious portion of the link to the site in HEX (or other encoding methods) so the request is less suspicious looking to the user when clicked on. After the data is collected by the web application, it creates an output page for the user containing the malicious data that was originally sent to it, but in a manner to make it appear as valid content from the website. Many popular guestbook and forum programs allow users to submit posts with html and javascript embedded in them. If for example I was logged in as "john" and read a message by "joe" that contained malicious javascript in it, then it may be possible for "joe" to hijack my session just by reading his bulletin board post. Further details on how attacks like this are accomplished via "cookie theft" are explained in detail below.

"What does XSS and CSS mean?"

Often people refer to Cross Site Scripting as CSS. There has been a lot of confusion with Cascading Style Sheets (CSS) and cross site scripting. Some security people refer to Cross Site Scripting as XSS. If you hear someone say "I found a XSS hole", they are talking about Cross Site Scripting for certain.

"What are the threats of Cross Site Scripting?"

Often attackers will inject JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable application to fool a user (Read below for further details) in order to gather data from them. Everything from account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible. New malicious uses are being found every day for XSS attacks. The post below by Brett Moore brings up a good point with regard to "Denial Of Service", and potential "auto-attacking" of hosts if a user simply reads a post on a message board.

"What can I do to protect myself as a vendor?"

This is a simple answer. Never trust user input and always filter metacharacters. This will eliminate the majority of XSS attacks. Converting < and > to < and > is also suggested when it comes to script output. Remember XSS holes can be damaging and costly to your business if abused. Often attackers will disclose these holes to the public, which can erode customer and public confidence in the security and privacy of your organization's site. Filtering < and > alone will not solve all cross site scripting attacks and it is suggested you also attempt to filter out (and) by translating them to (and);, and also # and & by translating them to # (#) and & (&).

"What can I do to protect myself as a user?"

The easiest way to protect yourself as a user is to only follow links from the main website you wish to view. If you visit one website and it links to CNN for example, instead of clicking on it visit CNN's main site and use its search engine to find the content. This will probably eliminate ninety percent of the problem. Sometimes XSS can be executed automatically when you open an email, email attachment, read a guestbook, or bulletin board post. If you plan on opening an email, or reading a post on a public board from a person you don't know BE CAREFUL. One of the best ways to protect yourself is to turn off Javascript in your browser settings. In IE turn your security settings to high. This can prevent cookie theft, and in general is a safer thing to do.

"How common are XSS holes?"

Cross site scripting holes are gaining popularity among hackers as easy holes to find in large websites. Websites from FBI.gov, CNN.com, Time.com, Ebay, Yahoo, Apple computer, Microsoft, Zdnet, Wired, and Newsbytes have all had one form or another of XSS bugs.

Every month roughly 10-25 XSS holes are found in commercial products and advisories are published explaining the threat.

"Does encryption protect me?"

Websites that use SSL (https) are in no way more protected than websites that are not encrypted. The web applications work the same way as before, except the attack is taking place in an encrypted connection. People often think that because they see the lock on their browser it means everything is secure. This just isn't the case.

"Can XSS holes allow command execution?"

XSS holes can allow Javascript insertion, which may allow for limited execution. If an attacker were to exploit a browser flaw (browser hole) it could then be possible to execute commands on the client's side. If command execution were possible it would only be possible on the client side. In simple terms XSS holes can be used to help exploit other holes that may exist in your browser.

"What if I don't feel like fixing a CSS/XSS Hole?"

By not fixing an XSS hole this could allow possible user account compromise in portions of your site as they get added or updated. Cross Site Scripting has been found in various large sites recently and have been widely publicized. Left unrepaired, someone may discover it and publish a warning about your company. This may damage your company's reputation, depicting it as being lax on security matters. This of course also sends the message to your clients that you aren't dealing with every problem that arises, which turns into a trust issue. If your client doesn't trust you why would they wish to do business with you?

Affected items

/search.asp

Details

The GET variable **tfSearch** has been set to **%3Cimg%20dynsrc%3D%22JaVaScRiPt:alert%28398177360470%29%3B%22%3E** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<DIV+STYLE="width:expression(alert(398157360470))%3B">** .

/search.asp

Details

The GET variable **tfSearch** has been set to **%3Cimg%20src%3D%22JaVaS%26%2399%3BRiPt:alert%28398127360437%29%3B%22%3E** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<iframe+src="data:text/html%3Bbase64,PHNjcmlwdD5hbGVydCgnYWN1bmV0aXgteHNzLXRlc3QnKTwwc2NyaXB0Pgo="+invalid="398197360470">** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<META+HTTP-EQUIV="refresh"+CONTENT="0%3Burl=JaVaS%26%2399%3BRiPt:alert(398187360470)%3B">** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<FRAMESET><FRAME+SRC="JaVaS%26%2399%3BRiPt:alert(398167360470)%3B"></FRAMESET>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **email@some<ScRiPt%20%0a%0d>alert(398107360437)%3B</ScRiPt>domain.com** .

/search.asp

Details

The GET variable **tfSearch** has been set to **>"><ScRiPt%20%0a%0d>alert(398067360437)%3B</ScRiPt>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<ScRiPt%20%0a%0d>alert(398047360437)%3B</ScRiPt>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **<script>alert(398037360437)</script>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **--><ScRiPt%20%0a%0d>alert(398097360437)%3B</ScRiPt>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **</title><ScRiPt%20%0a%0d>alert(398087360437)%3B</ScRiPt>** .

/search.asp

Details

The GET variable **tfSearch** has been set to **</textarea><ScRiPt%20%0a%0d>alert(398077360437)%3B</ScRiPt>** .

/search.asp
Details
The GET variable tfSearch has been set to <code></div><ScRiPt%20%0a%0d>alert(398297360503)%3B</ScRiPt></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><ScRiPt+bad=">" +src="http://testphp.onlinehackscan.com/xss.js?398277360503"></ScRiPt></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code>%3C/xss/*-*/style=xss:e/**/xpression(alert(398407360533))%3E</code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><iframe+/onload=alert(398437360563)></iframe></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code>%uff1e%uff02%uff1exxx%uff1cscript%uff1ealert(398387360532)%3B%uff1c/script%uff1e</code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><body+onload=alert(398217360470)></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><embed+src="http://testphp.onlinehackscan.com/xss.swf?398207360470"+type="application/x-shockwave-flash"></code>
/search.asp
Details
The GET variable tfSearch has been set to <code><ScRiPt+src=http://testphp.onlinehackscan.com/xss.js?398227360470></ScRiPt></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><img+src=http://testphp.onlinehackscan.com/dot.gif+onload=alert(398247360503)></code> .
/search.asp
Details
The GET variable tfSearch has been set to <code><script/xss+src=http://testphp.onlinehackscan.com/xss.js?398237360503></script></code> .

❗ Script source code disclosure

Severity	High
Type	Validation
Reported by module	Parameter manipulation

Description

It is possible to read the source code of this script by using script filename as a parameter. It seems that this script includes a file which name is determined using user-supplied data. This data is not properly validated before being passed to the include function.

Impact

An attacker can gather sensitive information (database connection strings, application logic) by analysing the source code.

This information can be used to launch further attacks.

Recommendation

Analyse the source code of this script and solve the problem.

Affected items

/templatize.asp

Details

The GET variable **item** has been set to **templatize.asp%00.html** .

/templatize.asp

Details

The GET variable **item** has been set to **templatize.asp%00.jpg** .

/templatize.asp

Details

The GET variable **item** has been set to **templatize.asp** .

🚫 SQL injection

Severity	High
Type	Validation
Reported by module	Parameter manipulation

Description

This script is possibly vulnerable to SQL Injection attacks.

SQL injection is a vulnerability that allows an attacker to alter backend SQL statements by manipulating the user input. An SQL injection occurs when web applications accept user input that is directly placed into a SQL statement and doesn't properly filter out dangerous characters.

This is one of the most common application layer attacks currently being used on the Internet. Despite the fact that it is relatively easy to protect against, there is a large number of web applications vulnerable.

Impact

An attacker may execute arbitrary SQL statements on the vulnerable system. This may compromise the integrity of your database and/or expose sensitive information.

Depending on the back-end database in use, SQL injection vulnerabilities lead to varying levels of data/system access for the attacker. It may be possible to not only manipulate existing queries, but to UNION in arbitrary data, use subselects, or append additional queries. In some cases, it may be possible to read in or write out to files, or to execute shell commands on the underlying operating system.

Certain SQL Servers such as Microsoft SQL Server contain stored and extended procedures (database server functions). If an attacker can obtain access to these procedures it may be possible to compromise the entire machine.

Recommendation

Your script should filter metacharacters from user input.
Check detailed information for more information about fixing this vulnerability.

Detailed information

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

SQL injection mitigations

We believe that web application developers often simply do not think about "surprise inputs", but security people do (including the bad guys), so there are three broad approaches that can be applied here.

Sanitize the input

It's absolutely vital to sanitize user inputs to insure that they do not contain dangerous codes, whether to the SQL server or to HTML itself. One's first idea is to strip out "bad stuff", such as quotes or semicolons or escapes, but this is a misguided attempt. Though it's easy to point out some dangerous characters, it's harder to point to all of them.

The language of the web is full of special characters and strange markup (including alternate ways of representing the same characters), and efforts to authoritatively identify all "bad stuff" are unlikely to be successful.

Instead, rather than "remove known bad data", it's better to "remove everything but known good data": this distinction is crucial. Since - in our example - an email address can contain only these characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
@. -_+
```

There is really no benefit in allowing characters that could not be valid, and rejecting them early - presumably with an error message - not only helps forestall SQL Injection, but also catches mere typos early rather than stores them into the database.

Be aware that "sanitizing the input" doesn't mean merely "remove the quotes", because even "regular" characters can be troublesome. In an example where an integer ID value is being compared against the user input (say, a numeric PIN):

```
SELECT fieldlist
  FROM table
 WHERE id = 23 OR 1=1;  -- Boom! Always matches!
```

In practice, however, this approach is highly limited because there are so few fields for which it's possible to outright exclude many of the dangerous characters. For "dates" or "email addresses" or "integers" it may have merit, but for any kind of real application, one simply cannot avoid the other mitigations.

Escape/Quotesafe the input

Even if one might be able to sanitize a phone number or email address, one cannot take this approach with a "name" field lest one wishes to exclude the likes of Bill O'Reilly from one's application: a quote is simply a valid character for this field.

One includes an actual single quote in an SQL string by putting two of them together, so this suggests the obvious - but wrong! - technique of preprocessing every string to replicate the single quotes:

```
SELECT fieldlist
  FROM customers
 WHERE name = 'Bill O''Reilly';  -- works OK
```

However, this naive approach can be beaten because most databases support other string escape mechanisms. MySQL, for instance, also permits `\` to escape a quote, so after input of `\'; DROP TABLE users; --` is "protected" by doubling the quotes, we get:

```
SELECT fieldlist
  FROM customers
 WHERE name = '\'; DROP TABLE users; --';  -- Boom!
```

The expression `\'` is a complete string (containing just one single quote), and the usual SQL shenanigans follow. It doesn't stop with backslashes either: there is Unicode, other encodings, and parsing oddities all hiding in the weeds to trip up the application designer.

Getting quotes right is notoriously difficult, which is why many database interface languages provide a function that does it for you. When the same internal code is used for "string quoting" and "string parsing", it's much more likely that the

process will be done properly and safely.

Some examples are the MySQL function `mysql_real_escape_string()` and perl DBD method `$dbh->quote($value)`. These methods must be used.

Use bound parameters (the PREPARE statement)

Though quotesafing is a good mechanism, we're still in the area of "considering user input as SQL", and a much better approach exists: bound parameters, which are supported by essentially all database programming interfaces. In this technique, an SQL statement string is created with placeholders - a question mark for each parameter - and it's compiled ("prepared", in SQL parlance) into an internal form. Later, this prepared query is "executed" with a list of parameters:

Example in perl

```
$sth->execute($email);
```

Thanks to Stefan Wagner, this demonstrates bound parameters in Java:

Insecure version

```
ResultSet rs = s.executeQuery("SELECT email FROM member WHERE name = "
    + formField); // *boom*
```

Secure version

```
"SELECT email FROM member WHERE name = ?");
ps.setString(1, formField);
ResultSet rs = ps.executeQuery();
```

Here, `$email` is the data obtained from the user's form, and it is passed as positional parameter #1 (the first question mark), and at no point do the contents of this variable have anything to do with SQL statement parsing. Quotes, semicolons, backslashes, SQL comment notation - none of this has any impact, because it's "just data". There simply is nothing to subvert, so the application is be largely immune to SQL injection attacks.

There also may be some performance benefits if this prepared query is reused multiple times (it only has to be parsed once), but this is minor compared to the enormous security benefits. This is probably the single most important step one can take to secure a web application.

Limit database permissions and segregate users

In the case at hand, we observed just two interactions that are made not in the context of a logged-in user: "log in" and "send me password". The web application ought to use a database connection with the most limited rights possible: query-only access to the members table, and no access to any other table.

The effect here is that even a "successful" SQL injection attack is going to have much more limited success. Here, we'd not have been able to do the UPDATE request that ultimately granted us access, so we'd have had to resort to other avenues.

Once the web application determined that a set of valid credentials had been passed via the login form, it would then switch that session to a database connection with more rights.

It should go almost without saying that sa rights should never be used for any web-based application.

Use stored procedures for database access

When the database server supports them, use stored procedures for performing access on the application's behalf, which can eliminate SQL entirely (assuming the stored procedures themselves are written properly).

By encapsulating the rules for a certain action - query, update, delete, etc. - into a single procedure, it can be tested and documented on a standalone basis and business rules enforced (for instance, the "add new order" procedure might reject that order if the customer were over his credit limit).

For simple queries this might be only a minor benefit, but as the operations become more complicated (or are used in more than one place), having a single definition for the operation means it's going to be more robust and easier to maintain.

Note: it's always possible to write a stored procedure that itself constructs a query dynamically: this provides no protection against SQL Injection - it's only proper binding with prepare/execute or direct SQL statements with bound variables that provide this protection.

Isolate the webserver

Even having taken all these mitigation steps, it's nevertheless still possible to miss something and leave the server open to compromise. One ought to design the network infrastructure to assume that the bad guy will have full administrator access to the machine, and then attempt to limit how that can be leveraged to compromise other things.

For instance, putting the machine in a DMZ with extremely limited pinholes "inside" the network means that even getting complete control of the webserver doesn't automatically grant full access to everything else. This won't stop everything, of course, but it makes it a lot harder.

Configure error reporting

The default error reporting for some frameworks includes developer debugging information, and this cannot be shown to outside users. Imagine how much easier a time it makes for an attacker if the full query is shown, pointing to the syntax error involved.

This information is useful to developers, but it should be restricted - if possible - to just internal users.

Affected items

/login.asp
Details
The POST variable tfUName has been set to ' .
/login.asp
Details
The POST variable tfUName has been set to onlinehackscan" .
/login.asp
Details
The POST variable tfUName has been set to \'
/login.asp
Details
The POST variable tfUPass has been set to onlinehackscan" .
/login.asp
Details
The POST variable tfUPass has been set to ' .
/login.asp
Details
The POST variable tfUPass has been set to \'
/register.asp
Details
The POST variable tfEmail has been set to onlinehackscan" .
/register.asp
Details
The POST variable tfEmail has been set to ' .
/register.asp
Details
The POST variable tfEmail has been set to \'

/register.asp

Details

The POST variable **tfRName** has been set to \'

/register.asp

Details

The POST variable **tfRName** has been set to ' .

/register.asp

Details

The POST variable **tfRName** has been set to **onlinehackscan''** .

/register.asp

Details

The POST variable **tfUName** has been set to ' .

/register.asp

Details

The POST variable **tfUName** has been set to \'

/register.asp

Details

The POST variable **tfUName** has been set to **onlinehackscan''** .

/register.asp

Details

The POST variable **tfUPass** has been set to ' .

/register.asp

Details

The POST variable **tfUPass** has been set to **onlinehackscan''** .

/register.asp

Details

The POST variable **tfUPass** has been set to \'

/search.asp

Details

The GET variable **tfSearch** has been set to \'

/search.asp

Details

The GET variable **tfSearch** has been set to **onlinehackscan''** .

/showforum.asp

Details

The GET variable **id** has been set to ' .

/showforum.asp

Details

The GET variable **id** has been set to **Jyl%3D** .

/showforum.asp

Details

The GET variable **id** has been set to \'

/showforum.asp

Details

The GET variable **id** has been set to **%2527** .

/showforum.asp
Details
The GET variable id has been set to onlinehackscan" .
/showthread.asp
Details
The GET variable id has been set to ' .
/showthread.asp
Details
The GET variable id has been set to onlinehackscan" .
/showthread.asp
Details
The GET variable id has been set to %2527 .
/showthread.asp
Details
The GET variable id has been set to JyI%3D .
/showthread.asp
Details
The GET variable id has been set to \' .

Cookie manipulation

Severity	Medium
Type	Validation
Reported by module	Parameter manipulation

Description

This script is vulnerable to Cookie manipulation attacks.

By injecting a custom HTTP header or by injecting a META tag, it is possible to alter the cookies stored in the browser. Attackers will normally manipulate cookie values to fraudulently authenticate themselves on a web site.

Impact

By exploiting this vulnerability, an attacker may conduct a session fixation attack. In a session fixation attack, the attacker fixes the user's session ID before the user even logs into the target server, thereby eliminating the need to obtain the user's session ID afterwards.

Recommendation

You need to filter the output in order to prevent the injection of custom HTTP headers or META tags. Additionally, with each login the application should provide a new session ID to the user.

Affected items

/search.asp
Details
The GET variable tfSearch has been set to <meta+http-equiv='Set-cookie'+content='cookievalue'> .

Application error message

Severity	Low
Type	Validation
Reported by module	Parameter manipulation

Description

This page contains an error/warning message that may disclose the sensitive information. The message can also contain the location of the file that produced the unhandled exception.

This may be a false positive if the error message is found in documentation pages.

Impact

The error messages may disclose sensitive information. This information can be used to launch further attacks.

Recommendation

Review the source code for this script.

Affected items

/login.asp

Details

The POST variable **tfUName** has been set to `\"");][*%0d%0a<%00` .

/login.asp

Details

The POST variable **tfUPass** has been set to `\"");][*%0d%0a<%00` .

/register.asp

Details

The POST variable **tfEmail** has been set to `0` .

/register.asp

Details

The POST variable **tfEmail** has been set to `\"");][*%0d%0a<%00` .

/register.asp

Details

The POST variable **tfEmail** has been set to `-268435455` .

/register.asp

Details

The POST variable **tfEmail** has been set to `0x3fffffff` .

/register.asp

Details

The POST variable **tfEmail** has been set to `NULL` .

/register.asp

Details

The POST variable **tfEmail** has been set to `0xffffffff` .

/register.asp

Details

The POST variable **tfEmail** has been set to `0x80000000` .

/register.asp

Details

The POST variable **tfEmail** has been set to `65536` .

/register.asp

Details

The POST variable **tfEmail** has been set to `-1.0` .

/register.asp

Details

The POST variable **tfEmail** has been set to `0x7fffffff` .

/register.asp

Details

The POST variable **tfEmail** has been set to **268435455** .

/register.asp

Details

The POST variable **tfEmail** has been set to .

/register.asp

Details

The POST variable **tfRName** has been set to **0xffffffff** .

/register.asp

Details

The POST variable **tfRName** has been set to **65536** .

/register.asp

Details

The POST variable **tfRName** has been set to **-1.0** .

/register.asp

Details

The POST variable **tfRName** has been set to **268435455** .

/register.asp

Details

The POST variable **tfRName** has been set to .

/register.asp

Details

The POST variable **tfRName** has been set to **\"");][*{0d%0a<%00** .

/register.asp

Details

The POST variable **tfRName** has been set to **-268435455** .

/register.asp

Details

The POST variable **tfRName** has been set to **0** .

/register.asp

Details

The POST variable **tfRName** has been set to **0x80000000** .

/register.asp

Details

The POST variable **tfRName** has been set to **0x7fffffff** .

/register.asp

Details

The POST variable **tfRName** has been set to **NULL** .

/register.asp

Details

The POST variable **tfRName** has been set to **0x3fffffff** .

/register.asp

Details

The POST variable **tfUName** has been set to **\"");][*{0d%0a<%00** .

/register.asp

Details

The POST variable **tfUPass** has been set to **0** .

/register.asp

Details

The POST variable **tfUPass** has been set to **NULL** .

/register.asp

Details

The POST variable **tfUPass** has been set to **\");][*%0d%0a<%00** .

/register.asp

Details

The POST variable **tfUPass** has been set to **268435455** .

/register.asp

Details

The POST variable **tfUPass** has been set to **-268435455** .

/register.asp

Details

The POST variable **tfUPass** has been set to **0x7fffffff** .

/register.asp

Details

The POST variable **tfUPass** has been set to **65536** .

/register.asp

Details

The POST variable **tfUPass** has been set to **-1.0** .

/register.asp

Details

The POST variable **tfUPass** has been set to **0x3fffffff** .

/register.asp

Details

The POST variable **tfUPass** has been set to **0x80000000** .

/register.asp

Details

The POST variable **tfUPass** has been set to **0xffffffff** .

/search.asp

Details

The GET variable **tfSearch** has been set to **\");][*%0d%0a<%00** .

/showforum.asp

Details

The GET variable **id** has been set to **268435455** .

/showforum.asp

Details

The GET variable **id** has been set to **-268435455** .

/showforum.asp

Details

The GET variable **id** has been set to **0x3fffffff** .

/showforum.asp

Details

The GET variable **id** has been set to **0x7fffffff** .

/showforum.asp

Details

The GET variable **id** has been set to **65536** .

/showforum.asp

Details

The GET variable **id** has been set to **-1.0** .

/showforum.asp

Details

The GET variable **id** has been set to **0xffffffff** .

/showforum.asp

Details

The GET variable **id** has been set to **NULL** .

/showforum.asp

Details

The GET variable **id** has been set to **\"");][*{%0d%0a<%00** .

/showforum.asp

Details

The GET variable **id** has been set to **0x80000000** .

/showthread.asp

Details

The GET variable **id** has been set to **0x3fffffff** .

/showthread.asp

Details

The GET variable **id** has been set to **65536** .

/showthread.asp

Details

The GET variable **id** has been set to **-1.0** .

/showthread.asp

Details

The GET variable **id** has been set to **0x7fffffff** .

/showthread.asp

Details

The GET variable **id** has been set to **NULL** .

/showthread.asp

Details

The GET variable **id** has been set to **268435455** .

/showthread.asp

Details

The GET variable **id** has been set to **-268435455** .

/showthread.asp

Details

The GET variable **id** has been set to **\"");][*{%0d%0a<%00** .

/showthread.asp
Details
The GET variable id has been set to 0xffffffff .
/showthread.asp
Details
The GET variable id has been set to 0x80000000 .
/templatize.asp
Details
The GET variable item has been set to -268435455 .
/templatize.asp
Details
The GET variable item has been set to 268435455 .
/templatize.asp
Details
The GET variable item has been set to .
/templatize.asp
Details
The GET variable item has been set to \");][*{%0d%0a<%00 .
/templatize.asp
Details
The GET variable item has been set to 0x7fffffff .
/templatize.asp
Details
The GET variable item has been set to 0xffffffff .
/templatize.asp
Details
The GET variable item has been set to 65536 .
/templatize.asp
Details
The GET variable item has been set to -1.0 .
/templatize.asp
Details
The GET variable item has been set to 0x80000000 .
/templatize.asp
Details
The GET variable item has been set to 0x3fffffff .
/templatize.asp
Details
The GET variable item has been set to NULL .
/templatize.asp
Details
The GET variable item has been set to 0 .

! Possible sensitive directories

Severity	Low
Type	Validation

Reported by module Directory checks

Description

A possible sensitive directory has been found. This directory is not directly linked from the website. This check looks for known sensitive directories like: backup directories, database dumps, administration pages, temporary directories. Each of those directories may help an attacker to learn more about his target.

Impact

This directory may expose sensitive information that may help an malicious user to prepare more advanced attacks.

Recommendation

Restrict access to this directory or remove it from the website.

Affected items

/cgi-bin

Details

No details are available.

/html

Details

No details are available.

User credentials are sent in clear text

Severity	Low
Type	Informational
Reported by module	Crawler

Description

User credentials are not encrypted when they are transmitted.

Impact

A third party may be able to read the user credentials by intercepting an unencrypted HTTP connection.

Recommendation

Because user credentials usually are considered sensitive information, it is recommended to be sent to the server over an encrypted connection.

Affected items

/login.asp

Details

It seems that user credentials are sent to **/login.asp in clear text**.

/login.asp

Details

It seems that user credentials are sent to **/login.asp in clear text**.

/login.asp

Details

It seems that user credentials are sent to **/login.asp in clear text**.

/login.asp

Details

It seems that user credentials are sent to **/login.asp in clear text**.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/login.asp

Details

It seems that user credentials are sent to [/login.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp

Details

It seems that user credentials are sent to [/register.asp](#) in clear text.

/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.
/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.
/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.
/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.
/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.
/register.asp
Details
It seemes that user credentials are sent to /register.asp in clear text.

GHDB: Typical login page

Severity	Informational
Type	Informational
Reported by module	GHDB - Google hacking database

Description

The description for this alert is contributed by the GHDB community, it may contain inappropriate language.

Category : [Pages containing login portals](#)

This is a typical login page. It has recently become a target for SQL injection. Comsec's article at <http://www.governmentsecurity.org/articles/SQLInjectionBasicTutorial.php> brought this to my attention.

The Google Hacking Database (GHDB) appears courtesy of the Google Hacking community.

Impact

Not available. Check description.

Recommendation

Not available. Check description.

Affected items

/login.asp
Details
We found inurl:login.asp

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp

Details

We found

[inurl:login.asp](#)

/login.asp
Details
We found inurl:login.asp
/login.asp
Details
We found inurl:login.asp
/login.asp
Details
We found inurl:login.asp
/login.asp
Details
We found inurl:login.asp
/login.asp
Details
We found inurl:login.asp

📘 Password type input with autocomplete enabled

Severity	Informational
Type	Informational
Reported by module	Crawler

Description

When a new name and password is entered in a form and the form is submitted, the browser asks if the password should be saved. Thereafter when the form is displayed, the name and password are filled in automatically or are completed as the name is entered. An attacker with local access could obtain the cleartext password from the browser cache.

Impact

Possible sensitive information disclosure

Recommendation

The password autocomplete should be disabled in sensitive applications.

To disable autocomplete, you may use a code similar to:
`<INPUT TYPE="password" AUTOCOMPLETE="off">`

Affected items

/login.asp
Details
Password type input named tfUPass from unnamed form with action login.asp has autocomplete enabled.
/login.asp
Details
Password type input named tfUPass from unnamed form with action GET RetURL=%2Ftemplize%2Easp%3Fitem%3Dhtml%2Fabout%2Ehtml has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fregister%2Easp%3F** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D1** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D1** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D0** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D2** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fsearch%2Easp%3F** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fsearch%2Easp%3FtfSearch%3D** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D2** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fdefault%2Easp%3F** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Ftemplatize%2Easp%3F** has autocomplete enabled.

/login.asp

Details

Password type input named **tfUPass** from **unnamed form** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D0** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Ftemplatize%2Easp%3Fitem%3Dhtml%2Fabout%2Ehtml** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D0** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Flogin%2Easp%3F** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fsearch%2Easp%3F** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fdefault%2Easp%3F** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fsearch%2Easp%3FtfSearch%3D** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **register.asp** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D2** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D0** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowforum%2Easp%3Fid%3D1** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Ftemplize%2Easp%3F** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D1** has autocomplete enabled.

/register.asp

Details

Password type input named **tfUPass** from form named **frmRegister** with action **GET RetURL=%2Fshowthread%2Easp%3Fid%3D2** has autocomplete enabled.