



SHALB Informational Security Agency

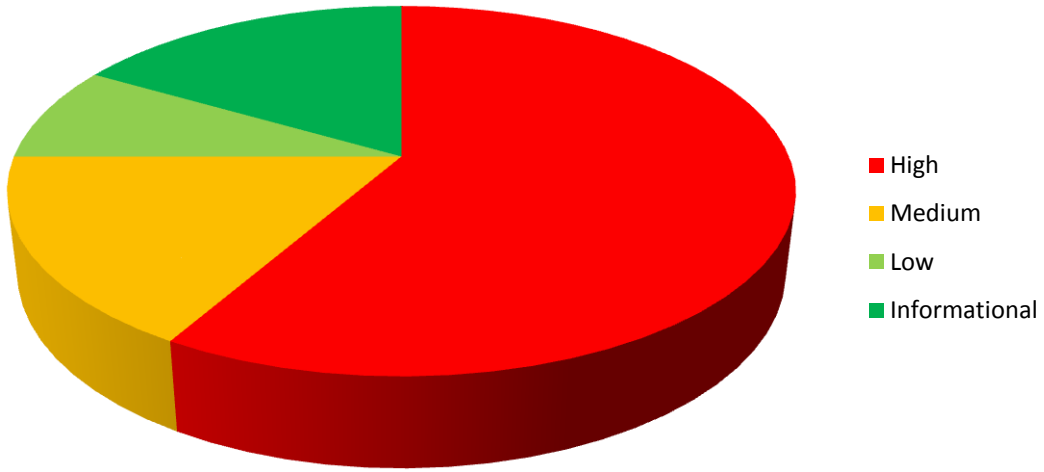
# Website Security Report

Sites: [test.shalb.com](http://test.shalb.com)

By SHALB Support

01/01/09

## Security vulnerabilities by risk



### Content:

<u>Alert group</u>	<u>Severity</u>	<u>Page</u>
Summary information	Informational	3
Apache version vulnerable	High	4
Cross Site Scripting	High	4
PHP code injection	High	7
PHP version vulnerable	High	8
SQL injection	High	10
Apache version vulnerable	Medium	14
Backup files	Medium	16
Possible sensitive directories	Low	16
Broken links	Informational	17

## Summary information

### Server information

Responsive	True
Server banner	Apache/2.0.55 (Ubuntu) mod_python/3.1.4 Python/2.4.3 PHP/5.1.2 mod_ssl/2.0.55 OpenSSL/0.9.8a mod_perl/2.0.2 Perl/v5.8.7
Server OS	Unix
Server	PHP,Perl,mod_ssl,mod_perl,mod_python,OpenSSL

### List of files with inputs

These files have at least one input (GET or POST):

- **/search.php** - 3 inputs
- **/artists.php** - 1 inputs
- **/cart.php** - 2 inputs
- **/guestbook.php** - 3 inputs
- **/userinfo.php** - 2 inputs
- **/showimage.php** - 2 inputs
- **/product.php** - 1 inputs
- **/listproducts.php** - 2 inputs
- **/secured/newuser.php** - 9 inputs
- **/secured/phpinfo.php** - 1 inputs
- **/comment.php** - 6 inputs

### Website Summary:

Site scripts are affected by multiply security flaws and it is possible provide DoS attack against website. Server side software is outdated. You have to apply all recommendations described in report.

To enhance web security we recommend deployment of web-firewall to prevent risk to be hacked using XSS, \*SQL Injections, Code Injections, and other vulnerabilities. Also we recommend to deploy network-based Intrusion Prevention Systemn to mitigate DoS attacks.



## Alert details

## Apache Mod\_Rewrite Off-By-One Buffer Overflow Vulnerability

Severity

High

**Web Server** Current version is **Apache/2.0.55**

### Description

*This alert was generated using only banner information. It may be a false positive.*

Apache mod\_rewrite is prone to an off-by-one buffer-overflow condition. The vulnerability arising in the mod\_rewrite module's ldap scheme handling allows for potential memory corruption when an attacker exploits certain rewrite rules.

Affected Apache versions:

- Apache 1.3.28 - 1.3.36 with mod\_rewrite
- Apache 2.2.0 - 2.2.2 with mod\_rewrite
- Apache 2.0.46 - 2.0.58 with mod\_rewrite

### Impact

An attacker may exploit this issue to trigger a denial-of-service condition. Reportedly, arbitrary code execution may also be possible.

### Recommendation

Upgrade Apache to the latest version.

## Cross Site Scripting

Severity

High

### Affected items

#### /comment.php

The POST variable **name** has been set to

**email@some<ScRiPt%20%0a%0d>alert(466290619302)%3B</ScRiPt>domain.com**

The POST variable **name** has been set to **1<script>alert(466220619302)</script>**

The POST variable **name** has been set to **1<iframe/+>onload=alert(466660620822)></iframe>**

The POST variable **name** has been set to

**1<img+src=http://test.shalb.com/dot.gif+onload=alert(466470620736)>**

#### /guestbook.php

The POST variable **name** has been set to **1<script>alert(423460513632)</script> .**

The POST variable **name** has been set to **1<iframe/+>onload=alert(424310514694)></iframe>**

The POST variable **text** has been set to

**1>'><ScRiPt%20%0a%0d>alert(424360515285)%3B</ScRiPt>**

The POST variable **text** has been set to **1<body+onload=alert(424520515431)>**

#### /listproducts.php

The GET variable **artist** has been set to `%3C/xss/*-`

`*/style=xss:e/**/xpression(alert(451070589307))%3E`

The GET variable **artist** has been set to `1--`

`><ScRiPt%20%0a%0d>alert(450310588062)%3B</ScRiPt>`

The GET variable **cat** has been set to

`1%00"><ScRiPt%20%0a%0d>alert(450660588663)%3B</ScRiPt>`

The GET variable **cat** has been set to

`1</title><ScRiPt%20%0a%0d>alert(450410588066)%3B</ScRiPt>`

### /search.php

The POST variable **searchFor** has been set to

`email@some<ScRiPt%20%0a%0d>alert(400700456928)%3B</ScRiPt>domain.com`

The POST variable **searchFor** has been set to

`1%00"><ScRiPt%20%0a%0d>alert(400940458760)%3B</ScRiPt>`

The POST variable **uname** has been set to

`1</title><ScRiPt%20%0a%0d>alert(456390603130)%3B</ScRiPt>`

### /secured/newuser.php

The POST variable **uname** has been set to `1<script>alert(456340603130)</script>`

The POST variable **uname** has been set to `1<iframe/+>onload=alert(456780603922)></iframe>`

## 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

### 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 &lt; and &gt; 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 &#40; and &#41;, and also # and & by translating them to &#35; (#) and &#38; (&).

**"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?

## PHP code injection

Severity

High

**Affected items****/comment.php**

The POST variable **phpaction** has been set to **printf(md5(shalb\_security\_test))%3Bexit%3B//**

The POST variable **phpaction** has been set to **printf(md5(shalb\_security\_test))%3Bexit%3B//**

**Description**

This script is vulnerable to PHP code injection.

PHP code injection is a vulnerability that allows an attacker to inject custom code into the server side scripting engine. This vulnerability occurs when an attacker can control all or part of an input string that is fed into an eval() function call. Eval will execute the argument as code.

**Impact**

Malicious users may inject PHP code which will be executed on the server side. It's possible to run system commands if the PHP interpreter allows system() or similar functions.

**Recommendation**

Your script should properly sanitize user input.

## PHP HTML Entity Encoder Heap Overflow Vulnerability

Severity

High

### Affected items

Current version is **PHP/5.1.2**

### Description

*This alert was generated using only banner information. It may be a false positive.*

Stefan Esser reported some vulnerabilities in PHP, which can be exploited by malicious people to cause a DoS (Denial of Service) or potentially compromise a vulnerable system. The vulnerabilities are caused due to boundary errors within the "htmlentities()" and "htmlspecialchars()" functions. If a PHP application uses these functions to process user-supplied input, this can be exploited to cause a heap-based buffer overflow by passing specially crafted data to the affected application. Successful exploitation may allow execution of arbitrary code, but requires that the UTF-8 character set is selected. For a detailed explanation of the vulnerability read the referenced article.

Vendor has released PHP 5.2.0 which fixes this issue.

Affected PHP versions (up to 4.4.4/5.1.6).

### Impact

Denial of service, remote code execution.

### Recommendation

Upgrade PHP to the latest version.

## PHP old version

Severity

High

### Affected items

Current version is **PHP/5.1.2**

### Description

*This alert was generated using only banner information. It may be a false positive.*

The PHP development team would like to announce the immediate availability of PHP 5.2.3. This release continues to improve the security and the stability of the 5.X branch as well as addressing two regressions introduced by the previous 5.2 releases. These regressions relate to the timeout handling over non-blocking SSL connections and the lack of HTTP\_RAW\_POST\_DATA in certain conditions. All users are encouraged to upgrade to this release.

### Security Enhancements and Fixes in PHP 5.2.3:

- Fixed an integer overflow inside chunk\_split() (by Gerhard Wagner, CVE-2007-2872)
- Fixed possible infinite loop in imagecreatefrompng. (by Xavier Roche, CVE-2007-2756)

- Fixed ext/filter Email Validation Vulnerability (MOPB-45 by Stefan Esser, CVE-2007-1900)
- Fixed bug #41492 (open\_basedir/safe\_mode bypass inside realpath()) (by bugs dot php dot net at chsc dot dk)
- Improved fix for CVE-2007-1887 to work with non-bundled sqlite2 lib.
- Added mysql\_set\_charset() to allow runtime altering of connection encoding.

The PHP development team would like to announce the immediate availability of PHP 5.2.5. This release focuses on improving the stability of the PHP 5.2.x branch with over 60 bug fixes, several of which are security related. All users of PHP are encouraged to upgrade to this release.

#### Security Enhancements and Fixes in PHP 5.2.5:

- Fixed dl() to only accept filenames. Reported by Laurent Gaffie.
- Fixed dl() to limit argument size to MAXPATHLEN (CVE-2007-4887). Reported by Laurent Gaffie.
- Fixed htmlentities/htmlspecialchars not to accept partial multibyte sequences. Reported by Rasmus Lerdorf
- Fixed possible triggering of buffer overflows inside glibc implementations of the fnmatch(), setlocale() and glob() functions. Reported by Laurent Gaffie.
- Fixed "mail.force\_extra\_parameters" php.ini directive not to be modifiable in .htaccess due to the security implications. Reported by SecurityReason.
- Fixed bug #42869 (automatic session id insertion adds sessions id to non-local forms).
- Fixed bug #41561 (Values set with php\_admin\_\* in httpd.conf can be overwritten with ini\_set()).

The PHP development team would like to announce the immediate availability of PHP 5.2.6. This release focuses on improving the stability of the PHP 5.2.x branch with over 120 bug fixes, several of which are security related. All users of PHP are encouraged to upgrade to this release.

#### Security Enhancements and Fixes in PHP 5.2.6:

- Fixed possible stack buffer overflow in the FastCGI SAPI identified by Andrei Nigmatulin.
- Fixed integer overflow in printf() identified by Maksymilian Aciemowicz.
- Fixed security issue detailed in CVE-2008-0599 identified by Ryan Permeh.
- Fixed a safe\_mode bypass in cURL identified by Maksymilian Arciemowicz.
- Properly address incomplete multibyte chars inside escapeshellcmd() identified by Stefan Esser.
- Upgraded bundled PCRE to version 7.6

#### Impact

Denial of service or ultimately arbitrary code execution.

#### Recommendation

Upgrade PHP to the latest version

## PHP Zend\_Hash\_Del\_Key\_Or\_Index vulnerability

Severity **High**

### Affected items

Current version is **PHP/5.1.2**

### Description

*This alert was generated using only banner information. It may be a false positive.*

Stefan Esser had discovered a weakness within the depths of the implementation of hashtables in the Zend Engine. This vulnerability affects a large number of PHP applications. It creates large new holes in many popular PHP applications. Additionally many old holes that were disclosed in the past were only fixed by using the unset() statement. Many of these holes are still open if the already existing exploits are changed by adding the correct numerical keys to survive the unset(). For a detailed explanation of the vulnerability read the referenced article.

Affected PHP versions (up to 4.4.2/5.1.3).

### Impact

Possible code execution, SQL injection, ...

### Recommendation

Upgrade PHP to the latest version.

## SQL injection

Severity **High**

### Affected items

#### /artists.php

The GET variable **artist** has been set to \'  
The GET variable **artist** has been set to %00'  
The GET variable **artist** has been set to \"

#### /listproducts.php

The HTTP header **accept-language** has been set to '  
The HTTP header **accept-language** has been set to \"  
The HTTP header **accept-language** has been set to Jyl=  
The GET variable **artist** has been set to %2527  
The GET variable **artist** has been set to \'  
The GET variable **cat** has been set to '  
The GET variable **cat** has been set to Jyl%3D  
The HTTP header **client-ip** has been set to \"  
The HTTP header **client-ip** has been set to '  
The HTTP header **referer** has been set to %27

The HTTP header **referer** has been set to \'  
The HTTP header **user-agent** has been set to \'  
The HTTP header **x-forwarded-for** has been set to '

### **/product.php**

The GET variable **pic** has been set to **%2527**  
The GET variable **pic** has been set to **%00'**

## **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**

### **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  
ABCDEFGHIJKLMNPOQRSTUVWXYZ  
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.

## Apache 2.x old version

Severity

Medium

### Affected items

Current version is **Apache/2.0.55**

### Description

Fixed in Apache httpd 2.0.61:

- **moderate** : mod\_proxy crash CVE-2007-3847

A flaw was found in the Apache HTTP Server mod\_proxy module. On sites where a reverse

proxy is configured, a remote attacker could send a carefully crafted request that would cause the Apache child process handling that request to crash. On sites where a forward proxy is configured, an attacker could cause a similar crash if a user could be persuaded to visit a malicious site using the proxy. This could lead to a denial of service if using a threaded Multi-Processing Module.

- **moderate** : mod\_status cross-site scripting CVE-2006-5752

A flaw was found in the mod\_status module. On sites where the server-status page is publicly accessible and ExtendedStatus is enabled this could lead to a cross-site scripting attack. Note that the server-status page is not enabled by default and it is best practice to not make this publicly available.

- **moderate** : Signals to arbitrary processes CVE-2007-3304

The Apache HTTP server did not verify that a process was an Apache child process before sending it signals. A local attacker with the ability to run scripts on the HTTP server could manipulate the scoreboard and cause arbitrary processes to be terminated which could lead to a denial of service.

- **moderate** : mod\_cache proxy DoS CVE-2007-1863

A bug was found in the mod\_cache module. On sites where caching is enabled, a remote attacker could send a carefully crafted request that would cause the Apache child process handling that request to crash. This could lead to a denial of service if using a threaded Multi-Processing Module.

Affected Apache versions (up to 2.0.60).

#### Fixed in Apache httpd 2.0.63:

- **low** : mod\_proxy\_ftp UTF-7 XSS CVE-2008-0005

A workaround was added in the mod\_proxy\_ftp module. On sites where mod\_proxy\_ftp is enabled and a forward proxy is configured, a cross-site scripting attack is possible against Web browsers which do not correctly derive the response character set following the rules in RFC 2616.

- **moderate** : mod\_status XSS CVE-2007-6388

A flaw was found in the mod\_status module. On sites where mod\_status is enabled and the status pages were publicly accessible, a cross-site scripting attack is possible. Note that the server-status page is not enabled by default and it is best practice to not make this publicly available.

- **moderate** : mod\_imap XSS CVE-2007-5000

A flaw was found in the mod\_imap module. On sites where mod\_imap is enabled and an imagemap file is publicly available, a cross-site scripting attack is possible.

Affected Apache versions (up to 2.0.62).

#### Impact

Check references for details about every vulnerability.

#### Recommendation

Upgrade Apache 2.x to the latest version.

## Backup files

---

Severity

Medium

### Affected items

`/index.bak`

### Description

A possible backup file has been found on your webserver. These files are usually created by developers to backup their work.

### Impact

Backup files can contain script sources, configuration files or other sensitive information that may help an malicious user to prepare more advanced attacks.

### Recommendation

Remove the file(s) if they are not required on your website. As an additional step, it is recommended to implement a security policy within your organization to disallow creation of backup files in directories accessible from the web.

## Possible sensitive directories

---

Severity

Low

### Affected items

`/admin``/secured`

### 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.

## Broken links

---

Severity **Informational**

### Affected items

[/ajax/index.php](#)

[/flash/add.swf](#)

[/privacy.php](#)

### Description

A broken link refers to any link that should take you to a document, image or webpage, that actually results in an error. This page was linked from the website but it is inaccessible.

### Impact

Problems navigating the site

### Recommendation

Remove the links to this file or make it accessible