

HealthCheck Vulnerability Report

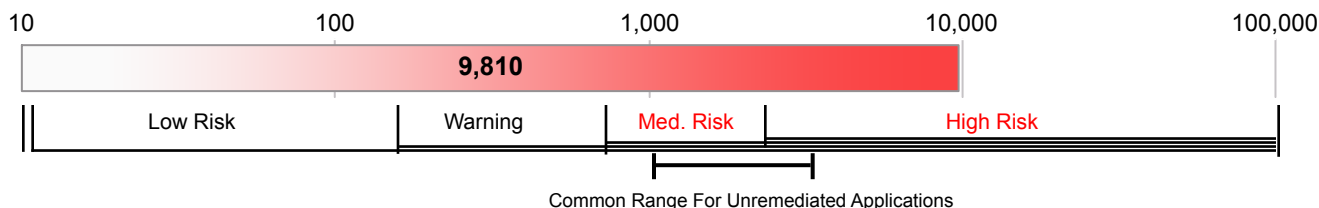
URL Assessed: <http://crackme.cenzic.com>

Summary

An assessment was performed on the target application, using 3 SmartAttacks™ to identify vulnerabilities. As shown below, a large number of issues were found with the application that resulted in a **very high HARM™ score of 9810**, which is commonly seen when there is at least one broadly repeated significant vulnerability. Typical HARM scores for complete assessments of individual non-remediated applications range from 1000 to 4500. **It is recommended that action be taken to remediate these vulnerabilities.** The tests performed, as well as specific vulnerabilities to be remediated, and their contribution to the total HARM score, can be viewed in the "SmartAttack Results (by HARM)" section of this report below.

[HARM Scoring](#)

Total HARM™ Score: 9810 = 9810 (Raw Scores) x 1.0 (App Risk Factor)

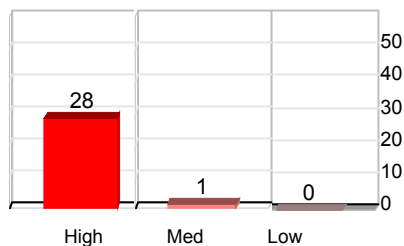


The 'Total HARM Score', above, is a sum of the HARM scores for all the SmartAttack assessments included in this report. SmartAttacks have different HARM scores based on the risks associated with each kind of vulnerability. The charts reflect the raw HARM scores without application specific risk adjustments.

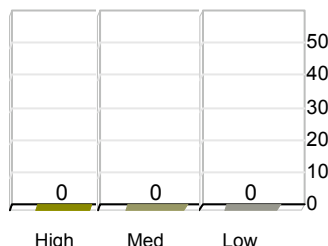
Severity of Findings

[Severity Drill Down](#)

[Severity Drill-down w/Info Items](#)



Vulnerabilities



Warnings

Pages Tested	93
Attack Count	5825

Note: **High, Med. & Low** relate to the severity of the findings. **Warnings** are findings for which there is less confidence of being real vulnerabilities.

High Severity

SmartAttack.	Vuln.	Warn.
■ Cross-Site Scripting	26	0
■ Non-SSL Password	2	0

Medium and Low Severity

SmartAttack.	Vuln.	Warn.
■ Password Autocomplete	1	0

SmartAttack Results (by HARM)[SmartAttack Drill-down](#)[SmartAttack Drill-down w/Info Items](#)

	<u>Status</u>	<u>Severity</u>	<u>HARM</u>	<u>Vuln</u>	<u>Warning</u>	<u>Info</u>
Cross-Site Scripting	Alert	High	8320	26	0	0
Non-SSL Password	Alert	High	1280	2	0	0
.....
Password Autocomplete	Note	Medium	210	1	0	0

Note: Detailed individual findings start on the next page.

Detail (by HARM / SmartAttack)

Attack: Cross-Site Scripting	Ver: 2.0.39	CWE-79	Observations: 26
Descr: Cross-site Scripting vulnerabilities allow malicious scripts to execute in the context of a trusted session with a web site. The SmartAttack alters the inputs to the web application to send benign versions of such malicious scripts, and detects the actual execution or unfiltered reflection of such scripts.			
Severity: High HARM: 320 Total HARM: 8320			

Impact

Cross-Site Scripting enables an attacker to run scripts inside a victim's browser. Using such a script, the attacker can modify the look-and-feel of a page, deface page contents and even steal user credentials and session information. If an application uses cookies for session management, then a Cross-Site Scripting vulnerability also assists the attacker in exploiting certain session-based attacks such as Session Fixation, if present.

Many Web applications display user input on their Web pages. Depending on whether the input is stored by the application for repeated use (e.g. user comments), a Cross-Site Scripting vulnerability may be *reflected* - i.e. usually one time - or *persistent*. A persistent Cross-Site Scripting vulnerability has greater impact than a reflected Cross-Site Scripting vulnerability, because a large number of users are affected without elaborate actions on the victims' part.

The effects of a Cross-Site Scripting vulnerability may range from simple defacement of Web pages to serious identity theft. For example, a Cross-Site Scripting vulnerability on a page that displays user-uploaded images could enable an attacker to show offensive images as if they were uploaded by a legitimate user, while a Cross-Site Scripting vulnerability on a banking Web site may expose the credentials of customers of the bank. While the offensive image may only affect a handful of users and the effect would be more annoyance than real harm, exposure of the credentials poses the threat of the attacker stealing money from them.

Cross-Site Scripting Vulnerable Findings

Assessment: HealthCheck, **Run:** 6/4/2011 4:50:13PM, **Traversal:** [Default Spider]

[Cross-Site Scripting](#)**1. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/register/register.php>**

Message: Cross-site scripting vulnerability found
Injected item: POST: UserId
Injection value: >'><script>alert(13072314.127)</script>
Detection value: 13072314.127
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

[Cross-Site Scripting](#)**2. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/view/updateloanrequest.php>**

Message: Cross-site scripting vulnerability found
Injected item: POST: txtFirstName
Injection value: >'><script>alert(13072314.5287)</script>
Detection value: 13072314.5287
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

[Cross-Site Scripting](#)**3. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/view/updateloanrequest.php>**

Message: Cross-site scripting vulnerability found
Injected item: POST: txtAnnualIncome
Injection value: >'><script>alert(13072314.7537)</script>
Detection value: 13072314.7537
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

[Cross-Site Scripting](#)**4. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/accttransaction.php>**

Message: Cross-site scripting vulnerability found
Injected item: POST: ToDate
Injection value: "><script>alert(13072314.12537)</script>

Detection value: 13072314.12537

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

5. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/accttransaction.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: FromDate
Injection value: "><script>alert(13072314.12587)</script>
Detection value: 13072314.12587

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

6. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.17717)</script>
Detection value: 13072314.17717

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

7. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.18537)</script>
Detection value: 13072314.18537

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

8. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.18977)</script>
Detection value: 13072314.18977

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

9. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.19867)</script>
Detection value: 13072314.19867

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

10. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.20687)</script>
Detection value: 13072314.20687

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

11. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.21527)</script>
Detection value: 13072314.21527

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

12. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.22377)</script>
Detection value: 13072314.22377

This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

13. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.23197)</script>
Detection value: 13072314.23197
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

14. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.24047)</script>
Detection value: 13072314.24047
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

15. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.24887)</script>
Detection value: 13072314.24887
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

16. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.25727)</script>
Detection value: 13072314.25727
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

17. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.26547)</script>
Detection value: 13072314.26547
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

18. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.27407)</script>
Detection value: 13072314.27407
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

19. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.28237)</script>
Detection value: 13072314.28237
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

20. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/loanrequestdetail.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserId
Injection value: "><script>alert(13072314.29077)</script>
Detection value: 13072314.29077
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

21. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/transfer.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: Amount
Injection value: "><script>alert(13072314.43827)</script>;//
Detection value: 13072314.43827
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

22. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/transfer.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: ToAccountNo
Injection value: "><script>alert(13072314.43817)</script>;//
Detection value: 13072314.43817
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

23. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/login.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserType
Injection value: "><script>alert(13072314.50647)</script>
Detection value: 13072314.50647
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

24. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/login.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hLoginType
Injection value: "><script>alert(13072314.50807)</script>
Detection value: 13072314.50807
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

25. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/login.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hUserType
Injection value: "><script>alert(13072314.53147)</script>
Detection value: 13072314.53147
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Cross-Site Scripting

26. Vulnerable (High, HARM: 320) at: <http://crackme.cenzic.com/Kelev/php/login.php>

Message: Cross-site scripting vulnerability found
Injected item: POST: hLoginType
Injection value: "><script>alert(13072314.53317)</script>
Detection value: 13072314.53317
This is a reflected XSS vulnerability, detected in an alert that was an immediate response to the injection.

Remediation Tips

The following general recommendations can help mitigate the risk associated with Cross-Site Scripting vulnerabilities. This is a complex problem area so there is no one simple fix or solution:

- Ensure that your web application validates all forms, headers, cookie fields, hidden fields, and parameters, and converts scripts and script tags to a non-executable form.
- Ensure that any executables on your server do not return scripts in executable form when passed scripts as malformed command parameters.
- Consider converting JavaScript and HTML tags into alternate HTML encodings (such as “<” to “<”).
- If your site runs online forums or message boards, disallow the use of HTML tags and Scripting in these areas.
- Keep up with the latest security vulnerabilities and bugs for all production applications and servers.
- Update your production servers with the latest XSS vulnerabilities by downloading current patches, and perform frequent security audits on all deployed applications.

The root cause of Cross-Site Scripting is a failure to filter hazardous characters from web application’s input and output. The two most critical programming practices you can institute to guard against Cross-Site Scripting are:

- Validate Input
- Encode output

Always filter data originating from outside your application by disallowing the use of special characters. Only display output to the browser that has been sufficiently encoded. When possible, avoid simple character filters and write routines that validate user input against a set of allowed, safe characters. Use regular expressions to confirm that data conforms to the allowed character set. This enhances application security and makes it harder to bypass input validation routines.

There are different tools you can use to validate and encode your data, depending upon your development environment. Your goal in Cross-Site Scripting attacks remediation should be to filter and encode all potentially dangerous characters so that the application does not return data that the browser will interpret as executable. Any non-escaped or non-encoded data that is returned to the browser is a potential security risk.

The following characters can be harmful and should be filtered whenever they appear in the application input or output. In output, you should translate these characters to their HTML equivalents before returning data to the browser.

> < () [] ' " ; : / |

PHP

The following PHP functions help mitigate Cross-Site Scripting **Vulnerabilities**:

Strip_tags() removes HTML and PHP scripting tags from a string.

Utf8_decode() converts UTF-8 encoding to single byte ASCII characters. Decoding Unicode input prior to filtering it can help you detect attacks that the attacker has obfuscated with Unicode encoding.

Htmlspecialchars() turns characters such as &, >, <,” into their HTML equivalents. Converting special characters to HTML prevents them from being executable within browsers when sent by an application.

Strtr() filters any characters you specify. Make sure to filter “; : ()” characters so that attackers cannot craft strings that generate alerts. Many XSS attacks are possible without the use of HTML characters, so filtering and encoding parentheses mitigates these attacks. For example:

```
" style="background:url(JavaScript:alert(Malicious Content));
```

ASP.NET

With ASP.NET, you can use the following functions to help prevent Cross-Site Scripting:

- Constrain input submitted via server controls by using ASP.NET validator controls, such as RegularExpressionValidator, RangeValidator, and System.Text.RegularExpressions.Regex. Using these methods as server-side controls to limit data input to only allowable character sequences by validating input type, length, format, and character range.
- Use the HtmlUtility.HtmlEncode method to encode data if it originates from either a user or from a database. HtmlEncode replaces special characters with their HTML equivalents, thus preventing the output from being executable in the browser. Use HtmlUtility.UrlEncode when writing URLs that may have originated from user input or stored database information.
- Use the HttpOnly cookie option for added protection.
- As a best practice, you should use regular expressions to constrain input to known safe characters. Do not rely solely on ASP.NET validateRequest, but use it in addition to your other input validation and encoding mechanisms.

Java

When it comes to writing some validation code, there are two main choices: filtering and encoding.

Vulnerable code:

Consider the following code:

```
<% String sid = request.getParameter("sid"); %>
```

...

```
Student ID: <%= sid %>
```

This code functions correctly when the values of name are as expected. Since there is no validation of this, things can go awry if the inputs are not as expected. For example, a javascript can be made to execute that steals cookies.

Remediation to prevent these problems is outlined below.

Secure code:

Filtering:

There are two types of filtering: positive and negative filtering.

Positive Filtering:

The safest and most prevalent method of preventing against attack is to only accept data that is valid and reject everything else. For example, if the data is expected to be alphanumeric, then any input that is not should be rejected.

```
String Str = request.getParameter("input");
```

```
String Pattern = "\\d+$";
```

```
if (!Str.matches(Pattern))
```

```
    /* invalid input, take appropriate action*/
```

Negative Filtering:

Even though this is the ideal mechanism, it might not be practical to reject all data. For example, in blogging applications, it might be a requirement to allow the user to use html format to input data. In this case, check for the existence of special characters within the data. These characters can be replaced with other characters, such as a space.

```
/* regular expression that
```

```
 * tests for the existence of malicious characters
```

```
 * and replaces them with a space.*/
```

```
String Pattern="[<>{}\\[\\];\\&]";
```

```
String Str = s.replaceAll(Pattern, " ");
```

Encoding:

To ensure that the generated pages are properly encoded for a Web server, use a simple mechanism rather than an application. Pass each character in the dynamic content through an encoding function where the scripting tags in the dynamic content are encoded.

A tag library is made up of one or more classes and an XML tag library description file, which dictates the new tag names and valid attributes for those tags. Tag handlers determine how the tags, their attributes, and their bodies are interpreted and processed at request time from inside a JSP page.

Examples of Encoding Functions are below:

```
public int Encode () throws Exception {
    StringBuffer sbuf = new StringBuffer();
    char[] chars = property.toCharArray();
    for (int i = 0; i < chars.length; i++)
        sbuf.append("&#" + (int) chars[i]);
    try
    {
        pageContext.getOut().print(sbuf.toString());
    } catch (IOException ex) {
        throw new JspException(ex.getMessage());
    }
    return;
}
```

Java HTML Encoding Function

```
public static String HTMLEncode(String aTagFragment){
    final StringBuffer result = new StringBuffer();
    final StringCharacterIterator iterator = new
        StringCharacterIterator(aTagFragment);
```

```

char character = iterator.current();
while (character != StringCharacterIterator.DONE )
{
    if (character == '<')
        result.append("&lt;");
    else if (character == '>')
        result.append("&gt;");
    else if (character == '\\"')
        result.append("&quot;");
    else if (character == '\\')
        result.append("&#039;");
    else if (character == '\\')
        result.append("&#092;");
    else if (character == '&')
        result.append("&amp;");
    else {
        //the char is not a special one
        //add it to the result as is
        result.append(character);
    }
    character = iterator.next();
}
return result.toString();
}

```

ColdFusion

Generic

ColdFusion provides a number of ways to filter or validate user input. The security measures span both client-side and server-side filtering. Where possible, implement your security controls on the server-side to avoid tampering of your controls via a Man-In-The-Middle (MITM) proxy. An attacker can easily bypass client-side controls by using such a program to modify the underlying HTTP Request as it passes between the proxy and the web application. Additionally, we recommend that sites using ColdFusion should configure their application using the recommended security features of the Adobe ColdFusion 8 Developers Guide, or the guide relevant to your version of ColdFusion. ColdFusion Data validation allows you to control the type of data that is allowed as well as to ensure that user-supplied data corresponds to the correct form. Attentive data validation procedures can have the following benefits:

- Enhance the security of your application by ensuring that malicious users cannot input data that exploits a security vulnerability, such as SQL Injection, XSS, or buffer overflows.
- Enhance application resilience by rejecting invalid data on the server-side prior to processing the input.
- Enhance application usability by providing the user with feedback that allows them to correct their mistakes, while not generating verbose error messages.

The list below gives you an overview of the available data validation tags, as well as their validation type (server vs. client side) methods. For a more detailed explanation consult your ColdFusion Developers resources on the CFML language and its security features:

- **Mask, client:** Applies to cinput tags on the client-side. The use of Mask creates a Javascript or ActionScript control that verifies that input corresponds to a specified pattern. For example: nnn-*n*-nnn-*n* where “*n*” is an integer. Note, this is a client-side control that can be easily bypassed.
- **onBlur, client:** Applies to cinput and ctextarea tags. *onBlur* creates a JavaScript that runs in the browser and checks that user supplied data matches a corresponding pattern. Can be bypassed by a MITM proxy.
- **onSubmit, client:** Applies to the Web browser when the user clicks submit. Checks that the data passed from the browser corresponds to a specified pattern. Can be bypassed by MITM proxy.
- **onServer, server:** Applies to server-side data after the form is submitted. ColdFusion checks the form data of cinput and ctextarea tags and generates an error page if the data is not valid. Use this tag in conjunction with the cferror tag to specify the validation error page. Note: a failure to specify an error page will result in an information leak in your error handling routine, as ColdFusion errors are verbose.

- *IsValid*, **server**: Tests an input variable to determine if the content of the variable meets internal validation rules. The *IsValid* function returns true or false for the variable.
- *Cfparam*, **server**: Tests an input variable to determine if the variable meets validation criteria. If the variable does not meet the criteria an expression exception is generated.
- *Cfqueryparam*, **server**: Evaluates the content of a HTTP query string to validate whether the string meets validation criteria. This tag is useful for scrubbing HTTP query strings prior to further processing.

ColdFusion Scriptprotect

In addition to the data validation techniques available in the CFML language ColdFusion also provides a Scriptprotect setting to further assist in the prevention of Cross-Site Scripting. The ColdFusion administrator should configure the Scriptprotect method in Application.cfm, under the Enable Global Script Protection setting. Using this setting will help to protect user input from Cross-Site Scripting, however, use of the global Scriptprotect method should not be a substitute for individual form and parameter validation techniques.

References

OWASP Frequently Asked Questions on Web Application Security: <http://www.owasp.org/documentation/appsecfaq>

Detection of SQL Injection and Cross-site Scripting Attacks: <http://www.securityfocus.com/infocus/1768>

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

CERT Advisory on Malicious HTML Tags: <http://www.cert.org/advisories/CA-2000-02.html>

Cross-Site Scripting Security Exposure Executive Summary:

<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/ExSumCS.asp>

Understanding the cause and effect of CSS Vulnerabilities: <http://www.technicalinfo.net/papers/CSS.html>

Remediation References

OWASP Guide to Building Secure Web Applications: <http://www.owasp.org/>

Preventing the Cross-Site Scripting Vulnerability: http://www.giac.org/practical/GSEC/Deyu_Hu_GSEC.pdf

CERT "Understanding Malicious Content Mitigation": http://www.cert.org/tech_tips/malicious_code_mitigation.html

OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation:

<http://www.owasp.org/documentation/guide/>

How to Build an HTTP Request Validation Engine (J2EE validation with Stinger):

<http://www.owasp.org/columns/jeffwilliams/jeffwilliams2>

Attack: Non-SSL Password	Ver: 1.2.6	CWE-201	Observations: 2
Descr: Non-SSL Password is a vulnerability caused by a failure to submit passwords via SSL. The SmartAttack inspects all requests where passwords are submitted and reports those where SSL is not used for the submission.			
Severity: High HARM: 640 Total HARM: 1280			

Impact

An attacker sniffing traffic on a computer network may notice and consequently steal usernames and passwords if they are sent unencrypted to a Web application. The attacker may then use this information to use the accounts corresponding to it for various malicious purposes, ranging from disclosure of personal information to identity theft.

Although the use of a packet sniffer or network monitoring tool is required to capture unencrypted traffic on a network, such tools are widely available and may be easy to install. This means that the complexity involved in capturing unencrypted traffic can be very less. Depending on which place the attacker is successfully able to install a sniffer, the scope of such an attack may be large, thereby increasing the chance of the attacker succeeding to get more passwords.

Non-SSL Password Vulnerable Findings

Assessment: HealthCheck, **Run:** 6/4/2011 4:50:13PM, **Traversal:** [Default Spider]

Non-SSL Password

1. Vulnerable (High, HARM: 640) at: <http://crackme.cenzic.com/Kelev/register/register.php>

Message: The action of a form containing a password field is not using SSL

Non-SSL Password

2. Vulnerable (High, HARM: 640) at: <http://crackme.cenzic.com/Kelev/php/changepass.php>

Message: The action of a form containing a password field is not using SSL

Remediation Tips

HTTPS uses Secure Socket Layer (SSL) encryption between client and server to protect data confidentiality. It is a recommended practice that any secure portion of your web application designed only for HTTPS access should be inaccessible via HTTP.

In addition to this, especially for login requests, it is highly recommended that requests containing username and password are sent over HTTPS, *i.e.* using SSL. To achieve this, ensure one of the following:

1. The username and password are being submitted to a URL which is explicitly an `https://` URL
2. If the username and password are being submitted to a URL relative to the URL of that page, then the page itself has an explicitly `https://` URL.

Attack: Password Autocomplete	Ver: 1.1.10	CWE-525	Observations: 1
Descr: Password Autocomplete is a vulnerability caused by allowing caching of passwords by browsers. The SmartAttack inspects responses from the application to identify if autocomplete is explicitly set to "off".			
Severity: Medium HARM: 210 Total HARM: 210			

Impact

Accessing a Web application with the Password Autocomplete vulnerability will cause users' passwords to be stored on the machine from where the browsing happens. An attacker who has access to such a machine may be able to make the browser give away those passwords by visiting the pages with the login forms, or by means of compromising the browser. He may be able to use those passwords for identity theft and/or performing malicious actions on behalf of the users.

An attacker who has access to a number of shared machines, such as at an Internet Cafe or posing as a computer maintenance professional may be able to collect passwords of users of applications with this vulnerability. It may also be possible for an attacker to install malware on victims' computers which will compromise the browser, thus making the attacker's presence at the machine unnecessary.

Password Autocomplete Vulnerable Findings

Assessment: HealthCheck, **Run:** 6/4/2011 4:50:13PM, **Traversal:** [Default Spider]

Password Autocomplete**1. Vulnerable (Medium, HARM: 210) at: <http://crackme.cenzic.com/Kelev/register/register.php>**

Message: Page contains password input fields without autocomplete='off': 1 forms.

```
<form method=post id=frm action=register.php name=frm >
<input size=30 value= tabindex=2 id=txtFirstName class=InputBox1 name=FirstName >
<input size=30 value= tabindex=2 id=txtLastName class=InputBox1 name=LastName >
<input size=30 value= tabindex=2 id=txtUserId class=InputBox1 name=UserId >
<input type=password maxlength=20 size=20 value= tabindex=2 id=txtpass class=InputBox1
name=Password >
<input maxlength=20 size=20 value= tabindex=2 id=txtDOB class=InputBox1 name=DOB >
<input size=49 value= tabindex=2 id=txtAddress class=InputBox1 name=Address >
<input size=30 value= tabindex=2 id=txtCity class=InputBox1 name=txtCity >
<input name=drpState class=InputBox1 >
<input maxlength=20 size=15 value= tabindex=2 id=txtTelephoneNo class=InputBox1
name=TelephoneNo >
<input maxlength=40 size=25 value= tabindex=2 id=txtEmail class=InputBox1 name=Email >
</form>
```

Remediation Tips

Turn off the AUTOCOMPLETE attribute in any HTML FORM/INPUT element that is used for passwords. This can be accomplished for a single field (such as a password field) by modifying the HTML source and adding the following line:

```
<INPUT TYPE = password NAME = value AUTOCOMPLETE = "off">
```

Alternately, you can use the <FORM AUTOCOMPLETE = "off"> attribute to disable autocomplete on every form field within a page.

If the page containing the password field is served over HTTPS and it was served with headers preventing caching of the data in the form, then Autocomplete is turned off for the form in Internet Explorer.

References

Microsoft Knowledge Base article, *Using AutoComplete in HTML Forms*

[http://msdn.microsoft.com/en-us/library/ms533032\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms533032(VS.85).aspx)

MSDN Library page on the Autocomplete attribute in HTML and CSS

[http://msdn.microsoft.com/en-us/library/ms533486\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms533486(VS.85).aspx)