

Assessment Report w/Remediation

Date Range: All

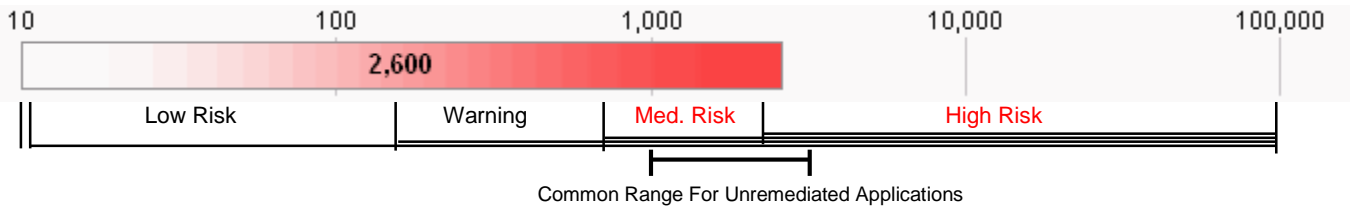
Application(s) Assessed: Hacked.NL

Summary

An assessment was performed on the Hacked.NL application, using 5 SmartAttacks™ to identify vulnerabilities. As shown below, several issues were found with the application that resulted in a **moderately high HARM™ score of 2600**. Typical HARM scores for complete assessments of individual non-remediated applications range from 1000 to 4500 (although large numbers of any one type of vulnerability can sometimes drive up scores to much higher levels). **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: 2600 = 2600 (Raw Scores) x 1.0 (App Risk Factor)

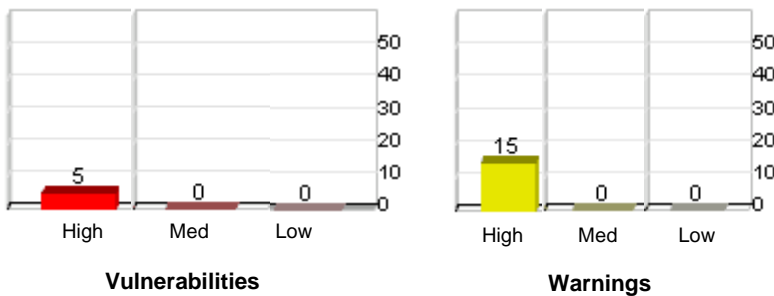


The 'Total HARM Score', above, is a sum of the HARM scores for all the SmartAttack assessments included in this report. SmartAttacks have different HARM scores based on the risks associated with each kind of vulnerability. The charts reflect the raw HARM scores without application specific risk adjustments.

Severity of Findings

[Severity Drill Down](#)

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



Pages Tested	150
Attack Count	13497
Informational Items	0

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

Medium and Low Severity

SmartAttack.	Vuln.	Warn.
■ Cross-Site Scripting	5	0
■ Form Caching	0	15

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>	<u>Attacks</u>	<u>Pages</u>	<u>Doc</u>
Cross-Site Scripting	Alert	High	1600	5	0	0	0	0	Doc
Form Caching	Alert	High	1000	0	15	0	0	0	Doc
Blind SQL Injection	Ok	High	0	0	0	0	0	0	Doc
SQL Disclosure	Ok	High	0	0	0	0	0	0	Doc
SQL Error Message	Ok	Medium	0	0	0	0	0	0	Doc

Detail (by SmartAttack / Report Item Type) [High Detail](#) [HighDetail w/Info Items](#) [\(Condensed Format\)](#)

Note: Detailed individual findings start on the next page.

Detail (by HARM / SmartAttack)

Attack: Form Caching	Ver: 1.1.13	CWE-525	Observations: 11
<p>Descr: A failure to specify proper caching directives may result into browsers caching HTML forms in their cache. This may reveal confidential data to an attacker. Hence, this is a vulnerability we call Form Caching. This SmartAttack reports each page, containing HTML forms, which does not have proper caching directives in its HTTP response.</p>			
Severity: High	Online Documentation for: Form Caching		

Impact

Accessing a Web application having a Form Caching vulnerability will cause content in HTML forms to be stored on the machine from which the browsing happens. An attacker who has access to such a machine may be able to make the browser give away those cached forms by visiting its cache. If the previously filled form entries contain sensitive information like Credit Card numbers, usernames *etc.* then the attacker can easily steal and misuse such information.

An attacker who has access to a number of shared machines, such as at an Internet Cafe or by posing as a computer maintenance professional, he may be able to collect sensitive data 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.

Form Caching Warning Findings

Assessment: Forus-P Default, **Run:** 6/3/2010 6:11:36PM, **Traversal:** Hacked.nl traversal_1

[Form Caching](#)**1. Warning (High, HARM: 192) at: <http://www.Hacked.nl/mijn-Hacked/registeren/>**

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =3
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >
<input type=text id=login-name class=text watermark name=login_name >
<input type=text id=login-password class=text watermark name=login_password >
<input type=checkbox name=login_remember >
/form>
<form method=get action=http://www.Hacked.nl/zoeken >
```

[Form Caching](#)**2. Warning (High, HARM: 192) at: <http://www.Hacked.nl/mijn-Hacked/registeren/>**

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =3
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >
<input type=text id=login-name class=text watermark name=login_name >
<input type=text id=login-password class=text watermark name=login_password >
<input type=checkbox name=login_remember >
/form>
<form method=get action=http://www.Hacked.nl/zoeken >
```

[Form Caching](#)**3. Warning (High, HARM: 192) at: <http://www.Hacked.nl/mijn-Hacked/registeren/>**

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =3
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >
<input type=text id=login-name class=text watermark name=login_name >
<input type=text id=login-password class=text watermark name=login_password >
<input type=checkbox name=login_remember >
```

```
/form>  
<form method=get action=http://www.Hacked.nl/zoeken >
```

Form Caching

4. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=43>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =4  
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >  
<input type=text id=login-name class=text watermark name=login_name >  
<input type=text id=login-password class=text watermark name=login_password >  
<input type=checkbox name=login_remember >  
/form>  
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

5. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=43>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =4  
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >  
<input type=text id=login-name class=text watermark name=login_name >  
<input type=text id=login-password class=text watermark name=login_password >  
<input type=checkbox name=login_remember >  
/form>  
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

6. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=43>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =4  
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >  
<input type=text id=login-name class=text watermark name=login_name >  
<input type=text id=login-password class=text watermark name=login_password >  
<input type=checkbox name=login_remember >  
/form>  
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

7. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=43>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =4  
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >  
<input type=text id=login-name class=text watermark name=login_name >  
<input type=text id=login-password class=text watermark name=login_password >  
<input type=checkbox name=login_remember >  
/form>  
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

8. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=45>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

```
Number of forms in the page cache =4  
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >  
<input type=text id=login-name class=text watermark name=login_name >  
<input type=text id=login-password class=text watermark name=login_password >  
<input type=checkbox name=login_remember >  
/form>
```

```
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

9. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=45>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

Number of forms in the page cache =4

```
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >
<input type=text id=login-name class=text watermark name=login_name >
<input type=text id=login-password class=text watermark name=login_password >
<input type=checkbox name=login_remember >
/form>
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

10. Warning (High, HARM: 192) at: <http://www.Hacked.nl/vraag-en-aanbod?cat=45>

Message: Only 'Cache-Control:no-Cache' found in the response header.It is recommended to use it along with 'Cache-Control:no-Store' to prevent caching.

Number of forms in the page cache =4

```
<form method=post action=http://www.Hacked.nl/mijn-Hacked/inloggen/ >
<input type=text id=login-name class=text watermark name=login_name >
<input type=text id=login-password class=text watermark name=login_password >
<input type=checkbox name=login_remember >
/form>
<form action=http://www.Hacked.nl/zoeken method=get >
```

Form Caching

11. Warning (High, HARM: 960) at: %3Curl_not_available%3E

Message: Limit exceeded. 5 more Warning item(s) were reported.
The total items reported individually have been limited to 10 items.
To change the number of reported items allowed,adjust the ReportItemLimit parameter.

Remediation Tips

This SmartAttack analyzes forms within the application to determine which forms allow content caching. When a form allows content to be cached, values previously entered into the form will be stored in the client's web browser. You can disable caching on a page by setting the "Pragma: No-cache" and "Cache-control: No-cache,No-Store" HTTP Header values. If you are using Active Server Pages, you can prevent the caching of form content with the following script:

```
<% Response.CacheControl = "no-cache,no-Store" %>
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = -1 %>
```

You can also set cache control in the HTML Header using META Tags as follows:

```
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache,no-Store">
```

NOTE: Be advised that if you use this method, pages might still be cached in Internet Explorer's temporary internet files. In order to resolve this issue, you must add an additional header to the page, as discussed in the Microsoft Knowledge Base article, "Pragma: No-cache" Tag May Not Prevent Page from Being Cached (<http://support.microsoft.com/default.aspx?kbid=222064>).

References

"Pragma: No-cache" Tag May Not Prevent Page from Being Cached
<http://support.microsoft.com/default.aspx?kbid=222064>:

Attack: Cross-Site Scripting	Ver: 2.0.28	CWE-79	Observations: 5
Descr: Cross-Site Scripting is a vulnerability caused by reflection of non-sanitized input which can be browser scripting statements in Web pages. This SmartAttack fault injects various Web forms and tries to detect a vulnerability by looking for alerts generated by its injections. It reports each page on which it finds Cross-Site Scripting vulnerabilities.			
Severity: High	Online Documentation for: Cross-Site Scripting		

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: [Forus-P Default](#), Run: [6/3/2010 6:11:36PM](#), Traversal: [Hacked.nl traversal_1](#)

[Cross-Site Scripting](#)

1. Vulnerable (High, HARM: 320) at: <http://www.Hacked.nl/zoeken?keyword=Passwor1&x=0&y=0>

Message: [Cross-site scripting vulnerability found](#)
Injected item: GET: keyword
Injection value: >><script>alert(12755828.7897)</script>
Detection value: 12755828.7897
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://www.Hacked.nl/zoeken?keyword=Passwor1&x=0&y=0&category=all&categories_advertising=zoek-iets&interests%255B%255D=3&city=Santa+Clara&age=25&constellation=0&is_extended_profile=1&forum_category=9&forum_author=testval

Message: [Cross-site scripting vulnerability found](#)
Injected item: GET: keyword
Injection value: >><script>alert(12755828.8867)</script>
Detection value: 12755828.8867
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://www.Hacked.nl/zoeken?keyword=Passwor1&x=0&y=0&category=all&categories_advertising=zoek-iets&interests%255B%255D=3&city=Santa+Clara&age=25&constellation=0&is_extended_profile=1&forum_category=9&forum_author=testval

Message: [Cross-site scripting vulnerability found](#)
Injected item: GET: category
Injection value: ">><script>alert(12755828.12167)</script>
Detection value: 12755828.12167
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://www.Hacked.nl/zoeken?category=advertising&keyword=Passwor1&x=0&y=0>

Message: Cross-site scripting vulnerability found
Injected item: GET: keyword
Injection value: ><<script>alert(12755828.22257)</script>
Detection value: 12755828.22257
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://www.Hacked.nl/zoeken?category=advertising&keyword=Passwor1&x=0&y=0>

Message: Cross-site scripting vulnerability found
Injected item: GET: category
Injection value: "><<script>alert(12755828.23347)</script>
Detection value: 12755828.23347
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.

Htmlespecialcharacters() 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.UriEncode` 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

akes, 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-*nnn*-*nnnn* where “n” is an integer. Note, this is a client-side control that can be easily bypassed.

onBlur, client: Applies to cinput and cftextarea 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 cftextarea 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>