

Vulnerability Advisory

Name	Multiple .NET Null Byte Injection Vulnerabilities
Vendor Website	http://www.microsoft.com
Date Released	July 11 th , 2007
Affected Software	.NET Framework 1.1, .NET Framework v2.0.50727
Researcher	Paul Craig : paul.craig@security-assessment.com

Description

Security-Assessment.com recently completed research into the .NET Framework in relation to the affect a Null byte (%00) has on various aspects of the .NET Common Language Runtime. This advisory details the findings of that research, conducted by Paul Craig paul.craig@security-assessment.com.

It was found that certain .NET methods in various sections of the .NET namespace are vulnerable to Null byte injection attacks. Null byte injection occurs when the .NET CLR incorrectly handles user supplied Null bytes.

The .NET CLR considers Null bytes as 'data', .NET strings are not Null byte terminated. However, native POSIX compliant function calls terminate all strings at the first found Null byte. Interoperability issues are encountered when data containing a Null byte is used by .NET to directly call a native C function call.

Native function calls terminate strings at the injected Null byte allowing a remote user to arbitrarily terminate a string parameter used by the vulnerable method.

Security-Assessment.com has discovered five vulnerable methods in the .NET framework which are exploited through Null byte injection. Three of the discovered vulnerabilities allow strings to be arbitrary terminated through String Termination vulnerabilities. The remaining two resulted in an Arbitrary File Disclosure condition where a remote user is capable of accessing arbitrary files from within the web root.

.NET has a history surrounding Null byte input flaws and associated logic. On September 8th, 2003 WebCohort Research research@webcohort.com released an advisory titled "Microsoft ASP.NET Request Validation Null Byte Filter Bypass Vulnerability". Use of a Null byte injection vulnerability allowed users to bypass the .NET request validation mechanism.

Null byte injection is not a new class of attack, and is a well known exploitive method but this is the first time a Null byte injection vulnerability has been found in methods within the .NET framework itself. Security researchers should be aware of Null byte injection attacks within the framework and within .NET developed applications.

Exploitation

The following examples can be found at <http://ha.cked.net/examples.zip>

Exploit 1: Server.MapPath

A Null byte injected within the filename parameter of the Server.MapPath method will terminate any returned string, removing any string data concatenated to the user supplied value. This can be seen in example 1 below.

```
name = test.aspx%00
```

```
Sub Page_Load()  
    dim name as string  
    dim realname as string  
    name = request("name") & ".uploaded"
```

```
realname = Mappath(".") & "\" & name
response.write("Mappath value of name variable: " & MapPath(name) & "<br>")
response.write("The real value is: " & realname & "<br>")
End Sub
```

Output:

Mappath of name variable = C:\inetpub\exploit1\test.aspx
The Real value is : C:\inetpub\exploit1\test.aspx.uploaded

1. Two variables are assigned, name and realname.
2. Name is a user supplied filename, and for security reasons .uploaded is appended to this value.
3. Real name is the virtual path for the current directory, with the name variable appended (This is the correct location of the file)
4. Inserting a Null byte suffix into the 'name' variable (name=test.aspx%00) is able to terminate the string returned from MapPath and any data concatenated to the user supplied value is removed.

Exploit 2: Server.Execute and Server.Transfer

Server.Execute and Server.Transfer were found to both be vulnerable to Null byte injection. Here the Null byte produces an arbitrary file disclosure vulnerability.

If a remote user is able to control the input used in a Server.Execute or Server.Transfer method, the method can be used to disclose the contents of any file within the document root.

As seen in example 2, below.

```
Sub Page_Load()
    Server.Transfer(request("page"))
End Sub
```

Security-Assessment.com has witnessed Server.Transfer and Server.Execute being used in page redirection functionality where .NET sessions are transferred to a user supplied page variable.

According to MSDN:

"The page transferred to should be another .aspx page. For instance, a transfer to an .asp or .asmx page is not valid. The Transfer method preserves the QueryString and Form collections."

If a user attempts to transfer to web.config, or any other none .ASPX page an exception is created. However, when the page variable is suffixed with a Null byte the complete file contents of the page is returned to the remote user.

A Server.Transfer or Server.Execute to page=web.config%00 will display the contents of the web.config file.

This vulnerability can be used to view any file within the document root.

Exploit 3: String.Compare

String.Compare was also found to be affected by Null bytes. Although does not produce an exploitable condition.

String.Compare demonstrates .NET's inability to correctly handle Null bytes.

This is demonstrated in the example below.

```
Sub Page_Load()  
    dim allowed, sFirstItem, sSecondItem as string  
    sFirstItem = Request("first")  
    sSecondItem = Request("second")  
    response.Write ("String.Compare - First item = " & sFirstItem & "<br>")  
    response.Write ("String.Compare - Second item = " & sSecondItem & "<br>")  
    if String.Compare(sFirstItem, sSecondItem) =0 then  
        response.Write ("<b>String.Compare - Matched! Strings are the same</b>" & "<br>")  
    else  
        response.Write ("<b>String.Compare - FAILED!! Strings are not the same</b>" &  
"<br>")  
    End If  
    if sFirstItem=sSecondItem then  
        response.Write ("Direct eval - Matched! Strings are the same" & "<br>")  
    else  
        response.Write ("<b>Direct eval - FAILED! Strings are not the same</b>" & "<br>")  
    End If  
End Sub
```

1. Two variables are supplied, "first" and "second"
2. String.Compare is called to compare the two strings, and then a direct evaluation is performed.
3. String.Compare should act identically to a direct evaluation.
4. The result of first=test&second=test can be seen below.

```
String.Compare - First item = test  
String.Compare - Second item = test  
String.Compare - Matched! Strings are the same  
Direct Eval - Matched! Strings are the same
```

This is the correct response, as both the first and second were defined as 'test'

5. Now, add a Null byte suffix to the 'first' variable and try again.

The result of Example3.aspx?first=test%00&second=test can be seen below.

```
String.Compare - First item = test  
String.Compare - Second item = test  
String.Compare - Matched! Strings are the same  
Direct Eval - Failed! Strings are not the same
```

String.Compare terminates the string at the first Null byte, and the two values are seen as identical. However, direct evaluation correctly determines that they are different, due to the extra Null byte suffix.

Although this is not exploitable it is an interesting finding.

Exploit 4: System.Net.Mail.SmtpMail.Send

The object System.Net.Mail.SmtpMail.Send was found to be vulnerable to string parameter termination similar to the vulnerability discovered in Server.MapPath.

This is demonstrated in the example below.

```
Private Sub Page_Load(sender As Object, e As System.EventArgs)  
    Dim m As New MailMessage()  
    m.From = "securityguy@security-assessment.com"  
    m.To = request("to") & "@security-assessment.com"  
    m.Subject = request("subject") & ": FromWebsite"
```

```
m.Body = request("body") & "This message was submitted by a user."
Response.Write("Sending mail to: " & m.to)
m.Fields.Add("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate", "1")
m.Fields.Add("http://schemas.microsoft.com/cdo/configuration/sendusername", "username")
m.Fields.Add("http://schemas.microsoft.com/cdo/configuration/sendpassword", "password")
SmtpMail.SmtpServer = "mail.server.com"
SmtpMail.Send(m)
```

End Sub

In this example the recipient property (To) is defined as a user supplied variable with "@security-assessment.com" concatenated. Security-Assessment.com has seen this development technique being used to ensure mail is only be sent to users of a particular domain.

However, if a user supplies a recipient (To) variable as paul.craig@microsoft.com%00 the mail will be sent to the Microsoft.com domain and string is again terminated at the first Null byte removing the concatenated "@security-assessment.com" value.

Similarly the From, Subject and Body values were found to be vulnerable to the same method of Null byte injection

Solution

- Security-Assessment.com has been in contact with Microsoft and new .NET patches have been released to address the discovered vulnerabilities.
- KB928365 (Security Update for Microsoft .NET Framework 2.0)
- KB928366 (Security Update For Microsoft .NET Framework 1.1)

About Security-Assessment.com

Security-Assessment.com is a leader in intrusion testing and security code review, and leads the world with SA-ISO, online ISO17799 compliance management solution. Security-Assessment.com is committed to security research and development, and its team have previously identified a number of vulnerabilities in public and private software vendors products.

For further information on this issue or any of our service offerings, contact us

Web www.security-assessment.com
Email info@security-assessment.com
Phone +649 302 5093