

# Web Server Discovery Tool

By Boris Loza, PhD, CISSP

## Introduction

This project started when I decided to find all the web servers on my network. One can do this by running `nmap` to identify all open HTTP/S related ports: 80, 8000, 8080, or 443. But `nmap` is known for crashing servers (just a couple of misbehaves to mention: killing `syslogd` on Solaris, Cisco's DOS, etc.) Therefore it is not allowed in some organizations. Moreover, even if the ports in question are open, `nmap` doesn't give you the type and the version of the web server listening to it. `Nmap` can also trigger the IDS and page the information security group! Using commercial tools like ISS Network Scanner or CyberCop to find all web servers on the network is cumbersome, time consuming, and IDS detectable.

Taking all this into consideration I decided to write my own tool for discovering all web servers on the network. I wanted this tool to be easy to run, not to use "crafted" TCP packets, be efficient, quick, and provide as much information about discovered web servers as possible. We intended to run this tool periodically, like a war dialer, and to do this even during business hours (before users shut down their workstations to go home). I wanted to create a tool as efficient as possible with minimum network and server impact. In this article you'll see what I eventually came up with.

## The Tool

First, let's understand a little bit about how a web server and a browser communicate. The browser or client generates request headers and sends them to the web server. The server receives the request headers, translates them, and generates the response headers. These response headers have to include information specific for the web server that will allow both the browser and the server to communicate. I decided to use this information to create the tool.

In the hart of the tool is the following Perl code:

```
1. use HTTP::Response;           #Encapsulate HTTP responses
2. use LWP::UserAgent;          #Dispatch WWW requests

3. my $ua = new LWP::UserAgent; #User agent object created
4. $ua->agent('Mozilla/5.0');    #Using Mozilla/5.0 as agent's name

5. my $req = new HTTP::Request(GET, "http://$ARGV[0]"); #Encapsulate a request using GET method
6. print $headers = $ua->request($req)->headers_as_string; #Read response from the web server
```

I use Perl's `libwww-perl` library for WWW access (rows 1 and 2). This library will provide the API for writing my own WWW clients.

First I need to create a request header (rows 3 and 4) by specifying the name of the web browser the request comes from. Now I can send the request to the server using the GET method (row 5). Strictly speaking, I can use any agent's name here, for example `agent('Foo')`. This doesn't matter,

since I need just one response from the server and I am not going to continue the session. Now I can print everything that comes from the server (row 6). After naming this little script as ws.pl and running it against one known web server I've got the following output:

```
C:\>ws.pl 192.168.0.40
Date: Thu, 04 Apr 2002 15:27:06 GMT
Accept-Ranges: bytes
Server: Microsoft-IIS/4.0
Content-Length: 56
Content-Location: http://192.168.0.40/Default.htm
Content-Type: text/html
ETag: "f82f8972cf9ac01:5ee8"
Last-Modified: Mon, 19 Feb 2001 23:55:33 GMT
Client-Date: Thu, 04 Apr 2002 15:28:43 GMT
Client-Peer: 192.168.0.40:80
X-Meta-Postinfo: /scripts/postinfo.asp
```

As I expected, the web server strikes back by sending all necessary information that will be needed for the session. If no HTTP web server is listening on port 80 the output will be:

```
C:\>ws.pl 10.56.53.27
Client-Date: Thu, 04 Apr 2002 18:38:39 GMT
```

In this article I am not going to explain all response headers from the output. For anybody who is interested, please refer to the RFC 2616. For the purpose of the script, I am interested only in one: Server: Microsoft-IIS/4.0. This is a name of the web server I connected to. So I can modify line 6 of the script to display only this response header

```
print $headers = $ua->request($req)->header('Server');
```

```
C:\>ws.pl 192.168.0.40
192.168.0.40 Microsoft-IIS/4.0
```

After understanding the concept, I started working on something more useful. Bellow is a listing of the complete tool. This tool will discover a single web server or all web servers on a given subnet. The default port to scan is 80, but you can specify any port you wish:

```
#Web Server Discovery Tool. Boris Loza, 2002
use HTTP::Response;
use LWP::UserAgent;
use Getopt::Std;
```

```
$usage="Use:\tws.pl [-v] [-p port] hostname
\tws.pl [-p port] -C IPaddress
\tws.pl -h {To print this}
```

```
Discover Web Servers.
Hostname can be specified by an IP address or a DNS name.
```

Options:

-v : verbose  
-p : specify a port (default 80)  
-C : scan class C subnet

Example: ws.pl -v 192.168.10.3 {OR}  
ws.pl myhost.com {OR}  
ws.pl -p 8000 myhost.com {OR}  
ws.pl -C 192.168.0 {OR}  
ws.pl -p 8000 -C 192.168.0";

```
getopts('C:hp:v') || die "$usage";
```

```
print "$usage" if $opt_h;
```

```
my $port=80; #Default port to scan  
if ($opt_p) {$port = $opt_p;}  
my $host = $ARGV[0];
```

```
#Create Request headers  
my $ua = new LWP::UserAgent;  
$ua->agent('Foo');
```

```
#Send Request headers  
my $req = new HTTP::Request(GET, "http://$host:$port");  
my $response = $ua->request($req);
```

```
#Use verbose mode. For single host only!  
if ($opt_v) {  
    print $response->headers_as_string;  
    exit;  
}
```

```
#Scan Class C Network  
$count = 1;  
if ($opt_C) {  
    (my $subnet, my $node) = ($opt_C =~ /(\d+\.\d+\.\d+)\.(\d+)/);  
    if ($node) {print $usage; exit;}  
    while ($count <=254) {  
        my $host = "$opt_C.$count";
```

```
#Skip unreachable hosts for speed (for Windows users only). Comment out for UNIX!  
if (^ ping $host` =~ m/(timed out)/) {$count++;next}
```

```
my $ua = new LWP::UserAgent;  
$ua->agent('Foo');  
my $req = new HTTP::Request(GET, "http://$host:$port");  
my $response = $ua->request($req);
```

```
if ($response->header('Server')) {  
    print $host,"t",$response->header('Server'),"n";  
} elsif ($response->header('Proxy-Agent')) {  
    print $host,"t",$response->header('Proxy-Agent'),"n";
```

```

    } elsif ($response->header('Title')) {
        print $host,"t",$response->header('Title'),"\n";
    } elsif ($response->header('Client-Peer')) {
        print $host,"t","Web Server not found, but port $port is open\n";
    }
    $count++;
}
exit;
}

if ($response->header('Server')) {
    print $ARGV[0],"t",$response->header('Server');
} elsif ($response->header('Proxy-Agent')) {
    print $ARGV[0],"t",$response->header('Proxy-Agent');
} elsif ($response->header('Title')) {
    print $ARGV[0],"t",$response->header('Title');
} elsif ($response->header('Client-Peer')) {
    print $ARGV[0],"t","Web Server not found, but port $port is open\n";
}

```

To run the ws.pl against a single host type:

```

C:\>ws.pl 192.168.0.2
10.56.49.2  Microsoft-IIS/4.0

```

Or specify a different port (port 80 is a default):

```

C:\>ws -p8000 192.168.0.58
192.168.0.58  HP-Web-Server-3.00.1696

```

You can use both an IP address and a DNS name here. For the verbose mode use the `-v` option. The following command will print all response headers for the host 192.168.0.2:

```

C:\>ws.pl -v 192.168.0.2
Date: Fri, 05 Apr 2002 15:49:09 GMT
Accept-Ranges: bytes
Server: Microsoft-IIS/4.0
Content-Length: 56
Content-Location: http://192.168.0.40/Default.htm
Content-Type: text/html
ETag: "f82f8972cf9ac01:5ee8"
Last-Modified: Mon, 19 Feb 2001 23:55:33 GMT
Client-Date: Fri, 05 Apr 2002 15:50:45 GMT
Client-Peer: 192.168.0.40:80
X-Meta-Postinfo: /scripts/postinfo.asp

```

To scan the whole class C network use the `-C` option. For example to discover all web servers running on port 80 on the subnet 192.168.0, type:

```

C:\>ws.pl -C 192.168.0
192.168.0.10  Microsoft-IIS/5.0

```

```
192.168.0.21    HTTP/1.0
192.168.0.33    IBM_HTTP_Server/1.3.6.4 Apache/1.3.7-dev (Unix)
192.168.0.34    IBM_HTTP_Server/1.3.12 Apache/1.3.12 (Unix)
192.168.0.40    Microsoft-IIS/4.0
192.168.0.45    Netscape-Enterprise/4.1
192.168.0.82    ApsyServer 1.0
192.168.0.90    Oracle HTTP Server Powered by Apache/1.3.12 (Win32) ApacheJServ/1.1 mod_ssl/2.6.4
                OpenSSL/0.9.5a mod_perl/1.24
192.168.0.91    Citrix Web PN Server
192.168.0.92    Web Server not found, but port 80 is open
```

HTTP/1.0 on the host 192.168.0.21 is a web interface for HP printer.

To scan the same subnet looking for web servers listening on port 8000, type:

```
C:\>ws.pl -p 8000 -C 192.168.0
.....
```

For help print ws.pl -h:

```
C:\>ws.pl -h
Use:  ws.pl [-v] [-p port] hostname
      ws.pl [-p port] -C IPaddress
```

Discover Web Servers.

Hostname can be specified by an IP address or a DNS name.

Options:

```
-v    : verbose
-p    : specify a port (default 80)
-C    : scan class C subnet
```

```
Example: ws.pl -v 192.168.10.3      {OR}
          ws.pl myhost.com          {OR}
          ws.pl -p 8000 myhost.com  {OR}
          ws.pl -C 192.168.0        {OR}
          ws.pl -p 8000 -C 192.168.0";
```

## Conclusion

After running this tool for the first time I found three times more web servers than I had on my list of the “official” web servers. The ws.pl has proved to be very efficient. Now I run it periodically to discover rogue web servers without any network or server impact. What else is important, I know what every line of this script is doing and can customize the script for my needs.