IOSMap: TCP and UDP Port Scanning on Cisco IOS Platforms

GCIA Gold Certification

Author  Robert VandenBrink
Advisor  Rick Wanner
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>The Business Requirement</td>
<td>3</td>
</tr>
<tr>
<td>Platform Selection and Caveats</td>
<td>4</td>
</tr>
<tr>
<td>Syntax Selection</td>
<td>5</td>
</tr>
<tr>
<td>Host Specification, Parsing IP’s and Ports</td>
<td>5</td>
</tr>
<tr>
<td>Validity Checks</td>
<td>5</td>
</tr>
<tr>
<td>Port Specification, Parsing Ports</td>
<td>6</td>
</tr>
<tr>
<td>Scan Types</td>
<td>6</td>
</tr>
<tr>
<td>TCP Connect Port Scanning</td>
<td>6</td>
</tr>
<tr>
<td>UDP Port Scanning</td>
<td>7</td>
</tr>
<tr>
<td>Ping Scanning</td>
<td>9</td>
</tr>
<tr>
<td>List Scan</td>
<td>9</td>
</tr>
<tr>
<td>Notes on Platform Impact and Change Control</td>
<td>9</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>10</td>
</tr>
<tr>
<td>Memory Utilization</td>
<td>11</td>
</tr>
<tr>
<td>Resource Utilization Watermarks</td>
<td>13</td>
</tr>
<tr>
<td>CPU Utilization</td>
<td>13</td>
</tr>
<tr>
<td>Running IOSmap (diagram and examples)</td>
<td>14</td>
</tr>
<tr>
<td>References</td>
<td>18</td>
</tr>
<tr>
<td>Appendix: Full Source Code Listing, Commented</td>
<td>19</td>
</tr>
</tbody>
</table>
Introduction

This paper describes IOSmap, a port scanning tool implemented on Cisco IOS using the native TCL (Tool Command Language) scripting language on that platform. The business requirement for this tool, implementation considerations and challenges, and design choices are discussed.

The Business Requirement

Writing a tool like a port scanner to run on Cisco IOS might seem like an unusual approach – some might say it sounds a lot like a solution looking for a problem. However, there are in fact some real-world scenarios where a solution of this type can fill a unique requirement.

One such circumstance is one that is sometimes seen as a security consultant, and involves some specific customer security constraints and requirements. For instance, suppose a company employee is suspected of running a peer-to-peer file distribution application. This violates Corporate Policy, and is illegal in the jurisdiction of the remote location in which the employee is working. The person being investigated is in the IT group, so has access and responsibility for change control on the corporate workstations and servers, and has responsibility for automated tools to report changes on these platforms. Further to this, the security access this employee regularly has is high enough to expose sensitive information, and the fear is that this information is exposed on the peer-to-peer network being used. The final constraint is that Corporate Policy further states that contractor hardware and non-approved software cannot be utilized in engagements of this type, for fear of inadvertent data exposure (via malware), or intentional data exposure.

This scenario in fact did occur, and the solution that was arrived at was to use a non-critical Cisco router (in fact, the router local to the network being investigated) to scan the suspect network for TCP ports commonly used by peer-to-peer file sharing applications. The port scan was scripted using TCL, a scripting language available on most modern routers and higher-end switches. After the initial port scans found suspect ports, the same local router was used to capture actual peer-to-peer traffic to build a body of evidence to take to the Corporate HR Group. After completing this engagement, the primitive port scanner was “cleaned up”, given some help text and transformed into a more general purpose tool (IOSmap) that can be used by others in similar circumstances.
Platform Selection and Caveats

Routing devices are almost always critical components of the infrastructure in any network, large or small. Because of this, it is recommended that wherever possible non-core, passive backup or spare hardware be used when running complex scripts. At all times, the resource constraints of the router should be kept in mind. Routers are typically constrained on memory, but often have CPU cycles to spare. Because of this, a short subroutine was written to estimate the overall impact of the script prior to execution. If the CPU or memory utilization is estimated to be potentially excessive, IOSmap displays an error message, and the user has an opportunity to exit the script immediately. Finally, TCL has no “Ctrl-C” equivalent on IOS, so larger scans that were perhaps started by accident are not easily ended (unless the terminal session is simply exited). For these reasons, IOSmap is most often used for “targeted scans”, where a limited number of addresses and/or ports are scanned. Full subnet scans or full range (1-65535) port scans are generally not recommended.

A discussion of operational caveats of tools of this type would not be complete without covering two more points: TCL requires privilege level 15 (full admin) rights to execute a script, and parts of IOSmap will modify the config and/or enable specific debugs. This should be kept in mind when using IOSmap, as change control requests will often be required for each of these 3 actions.
Application Syntax

The input syntax and output format was selected to be as close to standard, familiar tools as possible. To this end, both the inputs and outputs were designed to be similar to the popular NMAP scanning tool. The IOSmap tool is not presented as an NMAP port, it is a simple TCP and UDP port scanner on a constrained platform, so does not have either the capabilities, feature set, flexibility, breadth or speed of the NMAP tool.

All IOSmap parameters are defined at the command line. The help text for IOSmap shows all the scan options available:

```
HOST DISCOVERY:
-PO  Treat all hosts as online - skip Ping test
-SL  List hosts and ports to scan

SCAN TYPE:
-SP  Ping scan only <ICMP ECHO>
-ST  TCP Connect Scan
-SU  UDP Scan
--reason:  display the reason a port state is reported as such

PORT SPECIFICATION:
-p <port ranges> Specify ports to scan.
   -p22  Scan port 22
   -p22,23,135-139,445  Scan ports 22, 23, 135, 136, 137, 138, 139, 445

TARGET SPECIFICATION:
CIDR, IP range and single IPs are all a supported - comma delimited
For example:
192.168.10.0/24,192.168.17.21-34,192.168.40.1
```

Host Specification, Parsing IP’s and Ports

Validity Checks

Prior to execution, several checks are made to ensure that inputs are valid. All addresses specified are verified, to ensure that networks are specified with exactly 4 valid octets of 0-255, and that networks specified via CIDR notation are properly specified with bitmasks of 8-30. If the bitmask is less than 8, it is deemed that IOS not a good platform for the scan due to resource utilization, and the scan should be broken up if it was really required. If a bitmask greater than 30 is specified, individual addresses or a short address range is considered a better method to specify the target. Finally, networks or ports in ranges are verified that they are entered low-to-high.
Port Specification, Parsing Ports

Ports are entered as comma separated entities, specified with a "--p" command line argument. Each entity can be a single port, or a group of ports separated by a dash. For instance, the string "-p22-25,135,139" would specify ports 22, 23, 24, 25, 135 and 139. A validity check is done before proceeding to ensure that all ports are in the valid range of 1-65535 (scanning for port 0 is not supported).

Scan Types

Scan types can be either TCP, UDP, Ping only or List only.

TCP Connect Port Scanning

Since the TCL implemented on IOS does not permit the formation of raw packets, the only form of TCP scanning that can be realized is a simple TCP connect scan. If no scan type is specified on the IOSmap command line, TCP Scans are the default. TCP scanning can be implicitly specified with a "--sT" command line argument. The table below indicates the port status inferred for each possible return:

<table>
<thead>
<tr>
<th>Return code to TCP Connect request</th>
<th>Port Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Connect succeeds (three way TCP handshake completes)</td>
<td>Port is open</td>
</tr>
<tr>
<td>TCP Connect fails (three way TCP handshake does not complete)</td>
<td>Port is closed</td>
</tr>
</tbody>
</table>

A more complete table would look like:

<table>
<thead>
<tr>
<th>Packet returned</th>
<th>Port Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN/ACK</td>
<td>Port is open</td>
</tr>
<tr>
<td>RST from target</td>
<td>Port is closed</td>
</tr>
<tr>
<td>No response</td>
<td>There are 3 possible scenarios, and multiple checks to do in this case: If not on local network, <strong>port is filtered</strong> The host should return either an RST or ICMP packet (see below), or some intervening device should return an ICMP packet. If nothing is returned, a intervening firewall device is simply &quot;swallowing&quot; the packet. If on the local network, check ARP cache. If no arp</td>
</tr>
<tr>
<td>Entry</td>
<td>Result</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>RST from other ip address</td>
<td>Port is Filtered</td>
</tr>
<tr>
<td>ICMP Port Unreachable</td>
<td>Port is Filtered (see UDP Port Scanning Section)</td>
</tr>
<tr>
<td>ICMP Type 3, Code 3</td>
<td>Port is closed, as outlined in the UDP Port scanning section</td>
</tr>
<tr>
<td>All other ICMP Unreachables</td>
<td>Port is Filtered</td>
</tr>
</tbody>
</table>

This more complete table has not been implemented at this time, and are being considered for a future release.

### UDP Port Scanning

UDP port scanning is significantly more complex than TCP port scanning, especially on the IOS platform. Because there is no three-way handshake, UDP port scanning results must be inferred from other packet types that return when a UDP packet is sent to the port being tested. This means that a method of capturing “interesting packets” that return from a probe must be used. Finally, neither TCL nor the IOS command line has any method of generating a UDP packet.

Several methods were used to overcome these obstacles:

- UDP test packets are used by creating IP SLA’s to the test port. IP SLA’s are generally used to monitor performance of a particular port and/or protocol between two networks, especially if QOS and actual written service level agreements or requirements apply to intervening networks. Care is taken to ensure that SLA control packets are not used, as these “pollute” the output with UDP port 1967 control packets. Using SLA functions involves a configuration change to the routers’ running configuration. This means that UDP port scanning using this method should be subject to any change control procedures that govern the hardware platform being used.

- Return packets are captured by the router in a two step process. First an access list is created to define what an “interesting packet” might look like – we use access list 111, any ACL name or number might be used if this conflicts with the router configuration. Next, the local log in memory is cleared, and a “debug ip packet 111 detailed” is executed, which will capture the return packets to the log. After a short period of time (3 seconds minimum), the debug is stopped and the access list is removed. This method of packet capture has a few implications. First, debugs can take significant amounts of CPU. On modern hardware, this is normally not appreciable, but should be kept in mind. More importantly, this approach involves both a configuration change and a debug setting, both of which would require a change control request to be approved in most environments.
If used in a consulting engagement, even if change control is not of concern to the client, it might be a good idea to obtain written permission before running UDP scans in this way from an IOS platform.

The following table outlines the various cases that are tested for, and what the resulting port status is inferred to be. As can be seen, the majority of the feedback used to reach a decision is negative or null, it is rare to see actual UDP packets return from a request.

<table>
<thead>
<tr>
<th>ICMP Port Unreachable packet is returned</th>
<th>Port is considered closed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ICMP Type 3, Code 3, RFC792)</td>
<td>The ICMP Port Unreachable response comes from the target host, and indicates that it is not listening on this port</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Any other ICMP Destination Unreachable packet is returned</th>
<th>Port is considered “filtered”</th>
</tr>
</thead>
<tbody>
<tr>
<td>These include all ICMP Type 3 packets, with the following codes:</td>
<td>These packets are generally returned by network gear between the scanner and target, such as firewalls or routers. These indicate that the intervening gear is blocking the UDP probe packet with an ACL or other Firewall mechanism before it reaches the target host.</td>
</tr>
<tr>
<td>0  Net Unreachable [RFC792]</td>
<td></td>
</tr>
<tr>
<td>1  Host Unreachable [RFC792]</td>
<td></td>
</tr>
<tr>
<td>2  Protocol Unreachable [RFC792]</td>
<td></td>
</tr>
<tr>
<td>4  Fragmentation Needed and Don't Fragment was Set [RFC792]</td>
<td></td>
</tr>
<tr>
<td>5  Source Route Failed [RFC792]</td>
<td></td>
</tr>
<tr>
<td>6  Destination Network Unknown [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>7  Destination Host Unknown [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>8  Source Host Isolated [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>9  Communication with Destination Network is Administratively Prohibited [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>10 Communication with Destination Host is Administratively Prohibited [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>11 Destination Network Un reachable for Type of Service [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>12 Destination Host Unreachable for Type of Service [RFC1122]</td>
<td></td>
</tr>
<tr>
<td>13 Communication Administratively Prohibited [RFC1812]</td>
<td></td>
</tr>
<tr>
<td>14 Host Precedence Violation [RFC1812]</td>
<td></td>
</tr>
<tr>
<td>15 Precedence cutoff in effect [RFC1812]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UDP Packet returned from the target port</th>
<th>Port is considered to be “open”.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A return packet is a sure sign that the port is answering, however in most cases UDP ports do not return data when probed with zero-data packets.</td>
</tr>
</tbody>
</table>
Nothing is returned | Port is considered to be “open/filtered”.
This is the most frequent return if a port is open. Unfortunately, it is also what is returned on many firewalls if native IDS/IPS features are enabled. For this reason, an “all quiet” situation is generally inconclusive.

### Ping Scanning

Ping scanning is very straightforward. Target hosts are sent ICMP Echo Requests (pings), and hosts that return ICMP Echo Replies are considered to be up, and all other hosts are considered to be down. The native Cisco IOS “ping” command is called to execute the echo request.

### List Scan

A list scan simply lists the target addresses and ports that would be scanned. It is often used as a “preflight check” on the scan, to ensure that syntax is correct. It can also be used as input for a change control request, should one be required.

### Notes on Platform Impact and Change Control

As discussed, TCP scanning has a relatively low but measurable impact on the operational platform – since the “socket” command used in the TCP connect scan is part of the TCL language, no special measures are required to perform this function.

One thing to note in all scans is that IOS will view any scan run as “idle time”, as there is no keyboard activity during a scan. Ensure that the vty “exec-timeout” is long enough to accommodate extended scan runs, or they will be simply dropped when the vty session is terminated.

The main operational impact on the platform is memory utilization, which is easily quantified (more on this in the next section).

Similarly, the ping scan simply uses the native “ping” command in IOS in an exec call, so has minimal, quantifiable memory utilization, and requires no configuration changes. The List Scan has almost no impact at all, as it simply prints the scan targets to stdout.
However, UDP port scanning has several specific impacts.

- In order to send UDP packets, the running configuration is modified to create an IP SLA section. This is removed after each port scan is completed.
- Similarly, access list 111 is created to define “interesting” return traffic from a UDP scan, which is a second change to the running configuration.
- The “clear log” command simply doesn’t work in Cisco’s TCL implementation. To clear the log for each run, buffered logging is turned off then back on again – this achieves the exact same goal, but again is a running configuration change.
- All of these will create issues around approval of change control in a well run IT organization. In addition, these configuration changes will generate network alerts on many networks.
- Finally, the use of debugs in capturing the return traffic might also require approval under a change control process.

All of these issues will, on many networks, mean that UDP scanning is not practical with this tool.

**Resource Utilization**

Because routers are such critical devices, when adding new functions it is always important to gauge the impact of these new functions on performance in delivering their core functions. In particular, memory usage and CPU utilization are the two most important factors, both are easily measured. The memory usage of port scanning using TCL was especially interesting.
Memory Utilization

When scanning TCP ports, measuring memory allocation shows a fixed initial memory block used, then an almost straight line increase of memory usage as the total port count increases. Multiple IP’s do not contribute significantly to memory utilization, the critical factor is the total number of ports scanned.

<table>
<thead>
<tr>
<th>TCP Port Count Addresses</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2589644</td>
<td>2627088</td>
<td>2656128</td>
<td>2689332</td>
<td>2718372</td>
</tr>
<tr>
<td>2</td>
<td>2766588</td>
<td>2802504</td>
<td>2832996</td>
<td>2866200</td>
<td>2895240</td>
</tr>
<tr>
<td>3</td>
<td>2960112</td>
<td>2997480</td>
<td>3026520</td>
<td>3063888</td>
<td>3092928</td>
</tr>
<tr>
<td>4</td>
<td>3141144</td>
<td>3178512</td>
<td>3207552</td>
<td>3242208</td>
<td>3269796</td>
</tr>
</tbody>
</table>

TCP Port Scanning Memory Utilization

These can be represented closely (less than 0.5% error on each value)

**Memory = (IP’s * Ports * 1544) + 2568474**

Or, in more general terms:

**Memory = (Total TCP Ports) * 1544 + 2568474**

(Correlation Coefficient $R^2 = 0.999$, where $R^2 = 1$ is a perfect fit)
Memory Utilization when scanning UDP ports shows similarly linear behaviour.

<table>
<thead>
<tr>
<th>Addresses</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3021456</td>
<td>3429924</td>
<td>3762796</td>
<td>4141488</td>
<td>4515748</td>
<td>4884844</td>
<td>5259228</td>
<td>5632668</td>
<td>6002124</td>
</tr>
<tr>
<td>2</td>
<td>6667996</td>
<td>6962216</td>
<td>7252496</td>
<td>7537384</td>
<td>7827084</td>
<td>8120336</td>
<td>8410392</td>
<td>8695720</td>
<td>8984908</td>
</tr>
<tr>
<td>3</td>
<td>9570932</td>
<td>9856068</td>
<td>10145624</td>
<td>10439668</td>
<td>10729500</td>
<td>11014544</td>
<td>11303264</td>
<td>11597992</td>
<td>11892280</td>
</tr>
<tr>
<td>4</td>
<td>12470828</td>
<td>12764620</td>
<td>13054804</td>
<td>13339756</td>
<td>13628816</td>
<td>13922740</td>
<td>14207852</td>
<td>14497884</td>
<td>14787640</td>
</tr>
</tbody>
</table>

These can be represented closely by:

**Memory** = (IP’s * Ports * 30279) + 3120161

Or, in more general terms;

**Memory** = (Total UDP Ports) * 30279 + 3120161

(Correlation Coefficient R² = 0.999, where R² = 1 is a perfect fit)
The thing to note in this, is that the memory required for UDP port scanning is significantly higher than for TCP port scanning. Plotting both functions on the same graph shows this difference dramatically.

![Memory Utilization, TCP and UDP Port Scanning](image)

Memory Utilization, TCP and UDP Port Scanning

On a lightly loaded router with 256MB of DRAM, a UDP port scan of a full class C network for 35 ports will exceed the physical memory on the router. In a more realistic scenario, the Processor Pool Free Memory on such a router (3640, IP Plus feature set used as an example) will typically be in the 50MB range. A UDP port scan of a full class C network for 5 ports will exceed this value.

**Resource Utilization Watermarks**

If it is calculated that a given scan will exceed 50% of the available memory, the user is presented with a message and the opportunity to stop the scan. If it is calculated that a given scan will exceed 75% of the available memory, the scan is simply terminated with an error message.

**CPU Utilization**

CPU Utilization was uniformly less than 5% in all test TCP and UDP scans. This impact would be considered acceptable in most cases.
Running IOSmap (Network Diagram and Examples)

An example network (shown below) was constructed to demonstrate the use of the IOSmap tool:

In the example runs, IOSmap is called from a tftp path. This was done for simplicity, and is not recommended in a production environment for security reasons. Since the tftp protocol does not allow for authentication or encryption, a tftp server can easily be compromised, and TCL scripts stored on it could easily be updated to include rootkits or other “malcode”. Using a man-in-the-middle attack, TCL scripts could simply be intercepted and similarly modified in transit between the tftp server and client, without targeting anything on the tftp server at all. In a production situation, an SCP server would generally be recommended, as it provides for both encryption and authentication.
Sample IOSmap runs are shown below:

Scan 1 – a TCP scan of targeted hosts and ports

```
R#tclsh tftp://sec503/iosmap.tcl 1.1.1.1-5,2.2.2.5 -p7-9,13,19,22-24,80,443
Loading iosmap.tcl from 192.168.206.1 (via FastEthernet0/0): !
[OK - 14830 bytes]
Loading services.list from 192.168.206.1 (via FastEthernet0/0): !
[OK - 42121 bytes]
Starting IOSmap 0.9 ( http://www.defaultroute.ca ) at 2002-03-01 18:18 UTC
Free Memory on Platform = 47298216 / Memory required for this scan = 2661114

Interesting ports on host 1.1.1.1
PORT     STATE      SERVICE
7/tcp   closed     echo
8/tcp   closed     .
9/tcp   closed     discard
13/tcp   closed     daytime
19/tcp   closed     chargen
22/tcp   open       ssh
23/tcp   open       telnet
24/tcp   closed     priv-mail
80/tcp   open       http
443/tcp   open       https

Interesting ports on host 1.1.1.2
PORT     STATE      SERVICE
7/tcp   closed     echo
8/tcp   closed     .
9/tcp   closed     discard
13/tcp   closed     daytime
19/tcp   closed     chargen
22/tcp   open       ssh
23/tcp   open       telnet
24/tcp   closed     priv-mail
80/tcp   open       http
443/tcp   open       https

Interesting ports on host 1.1.1.3
PORT     STATE      SERVICE
7/tcp   open       echo
8/tcp   closed     .
9/tcp   open       discard
13/tcp   open      daytime
19/tcp   open       chargen
22/tcp   open       ssh
23/tcp   open       telnet
24/tcp   closed     priv-mail
80/tcp   open       http
443/tcp   open       https

Interesting ports on host 1.1.1.4
PORT     STATE      SERVICE
```
Interesting ports on host 1.1.1.5 is unavailable

Interesting ports on host 2.2.2.5
PORT     STATE      SERVICE
7/tcp   closed     echo
8/tcp   closed     .
9/tcp   closed     discard
13/tcp  closed     daytime
19/tcp  closed     chargen
22/tcp  open       ssh
23/tcp  closed     telnet
24/tcp  closed     priv-mail
80/tcp  open       http
443/tcp  closed     https

Scan 2 – a UDP scan of targeted hosts and ports

R1# tclsh tftp://sec503/iosmap.tcl 1.1.1.1-5,2.2.2.5 -p7-9,68-69,123 -sU
Loading iosmap.tcl from 192.168.206.1 (via FastEthernet0/0): !
[OK - 14830 bytes]
Loading services.list from 192.168.206.1 (via FastEthernet0/0): !
[OK - 42121 bytes]
Starting IOSmap 0.9 ( http://www.defaultroute.ca ) at 2002-03-01 18:21 UTC
Free Memory on Platform = 47293508 / Memory required for this scan = 4210205

Interesting ports on host 1.1.1.1
PORT     STATE      SERVICE
7/udp   closed     echo
8/udp   closed     .
9/udp   closed     discard
68/udp  closed     dhcpc
69/udp  closed     tftp
123/udp  open     ntp

Interesting ports on host 1.1.1.2
PORT     STATE      SERVICE
7/udp   closed     echo
8/udp   closed     .
9/udp   closed     discard
68/udp  closed     dhcpc
69/udp  open      tftp
123/udp open     ntp

Interesting ports on host 1.1.1.3

28 Sept, 2008
Robert VandenBrink
Page 16 of 27
<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/udp</td>
<td>open</td>
<td>echo</td>
</tr>
<tr>
<td>8/udp</td>
<td>closed</td>
<td></td>
</tr>
<tr>
<td>9/udp</td>
<td>open</td>
<td>discard</td>
</tr>
<tr>
<td>68/udp</td>
<td>closed</td>
<td>dhcpc</td>
</tr>
<tr>
<td>69/udp</td>
<td>closed</td>
<td>tftp</td>
</tr>
<tr>
<td>123/udp</td>
<td>open</td>
<td>ntp</td>
</tr>
</tbody>
</table>

Interesting ports on host 1.1.1.4

<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/udp</td>
<td>closed</td>
<td>echo</td>
</tr>
<tr>
<td>8/udp</td>
<td>closed</td>
<td></td>
</tr>
<tr>
<td>9/udp</td>
<td>closed</td>
<td>discard</td>
</tr>
<tr>
<td>68/udp</td>
<td>closed</td>
<td>dhcpc</td>
</tr>
<tr>
<td>69/udp</td>
<td>open</td>
<td>tftp</td>
</tr>
<tr>
<td>123/udp</td>
<td>open</td>
<td>ntp</td>
</tr>
</tbody>
</table>

Host 1.1.1.5 is unavailable

Interesting ports on host 2.2.2.5

<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/udp</td>
<td>closed</td>
<td>echo</td>
</tr>
<tr>
<td>8/udp</td>
<td>closed</td>
<td></td>
</tr>
<tr>
<td>9/udp</td>
<td>closed</td>
<td>discard</td>
</tr>
<tr>
<td>68/udp</td>
<td>closed</td>
<td>dhcpc</td>
</tr>
<tr>
<td>69/udp</td>
<td>open</td>
<td>tftp</td>
</tr>
<tr>
<td>123/udp</td>
<td>open</td>
<td>ntp</td>
</tr>
</tbody>
</table>
References


Appendix: Full Source Code Listing, Commented

```tcl
# set defaults if not over-ridden at cmd line
set pingit 1          ; # ping scan set to no (should be yes)
set scantype T        ; # default scan is TCP
set ports ""          ; # default scan is TCP
set portlist 0        ; # default scan is TCP
set timeout 1          ; # default scan is TCP
set timeouts 500       ; # default scan is TCP
set waittime 4000      ; # default scan is TCP
set svcarraymax 4096   ; # default scan is TCP
set fullpath $argv0    ; # default scan is TCP
set reasoncode 0       ; # default scan is TCP
set reason "NULL"      ; # default scan is TCP

This application consists of many procedures (procs)
These are at the beginning of the listing, the main executable
Is at the bottom of this listing

proc setloadpath { fullpath } {
    # get the path that IOSmap was called from
    global loadpath
    set firstchar 0
    set endchar [expr [string last "/" $fullpath ] -1 ]
    set loadpath [string range $fullpath $firstchar $endchar ]
    return $loadpath
}

This helptext subroutine prints all the cmd line syntax available
puts stdout 
"7=".................................................................
7 prints an ASCII 7 character (aka "bell"). This emits a "beep" when
printed to STDOUT

puts stdout  "IOScan 0.1"
puts stdout  "  Usage: IOScan <Scan Type> <Options> <target
specifications>

puts stdout  "HOST DISCOVERY:"
puts stdout  "  -P0  Treat all hosts as online - skip Ping test"
puts stdout  "  -SL  List hosts and ports to scan"
puts stdout  "SCAN TYPE:"
puts stdout  "  -sP  Ping scan only <ICMP ECHO>"
puts stdout  "  -sT  TCP Connect Scan"
puts stdout  "  -sU  UDP Scan"
puts stdout  "PORT SPECIFICATION:"
puts stdout  "  -p <port ranges> Specify ports to scan. "
puts stdout  "  -p22  Scan port 22"
puts stdout  "  -p22,23,135-139,445  Scan ports 22, 23, 135, 136, 137,
  138, 139, 445"
puts stdout  "TARGET SPECIFICATION:"
puts stdout  "  -c I, IP range and single IPs are all a supported - comma
delimited"
puts stdout  "  For example:"
puts stdout  "  192.168.10.0/24,192.168.17.21-34,192.168.40.1"
}

proc memcalc { scantype } {
    # this procedure calculates the estimated memory usage the scan run
    # will take.
    global iplist
    The iplist is the list of all ip's to be scanned
```
The portlist is the list of ports to be scanned

If the scan is for TCP ports, define the equation characteristics

- set gradient 1544
- set intercept 2568474

Similarly, these are the characteristics for a UDP scan.

- set gradient 30279
- set intercept 3120161

These are our safety factors (50 and 75%)

- set factor1 50 ; # watermark to ask for a y/n to proceed
- set factor2 75 ; # watermark to force an exit
- set ipcount [ length $iplist ]
- set portcount [ length $portlist ]
- set calcmem [ expr ($portcount * $ipcount * $gradient) ]
- set calcmem [ expr ( $calcmem + $intercept) ]
- set i [ exec "sho proc mem | i Processor Pool" ]
- set memfree [ index $i [ expr ( [llength $i] - 1 )] ]
- set memlimit1 [ expr ($memfree / 100 * $factor1) ]
- set memlimit2 [ expr ($memfree / 100 * $factor2) ]

Print memory values – this keeps it top-of-mind for anyone running the script

- puts stdout "Free Memory on Platform = $memfree / Memory required for this scan = $calcmem"

We’re over 75% - exit the application

- puts stdout "The resources used for your scan will exceed $factor2%" of your available memory total of $calcmem"
- puts stdout "Execution cannot proceed without impacting primary device functions"
- return 1

Memory usage calc is over 50% - ask for confirmation to proceed.

- puts stdout "The resources used by your scan will exceed the physical memory installed"
- puts stdout "of your available memory total of $calcmem"
- puts stdout "This may impact primary device functions"
- puts -nonewline stdout "do you wish to proceed (y/n) ==> ": flush stdout
- set response [ gets stdin ];
- if { $response == "y"} { return 0 } else { return 1 }
- return 0

Final case, all is well, proceed with the scan.

Convert an ip address to it's hexadecimal equivalent

- proc IPtoHex { IP } { binary scan [binary format c4 [split $IP .]] H8 Hex
- return $Hex
- }

Convert a hexadecimal value to decimal

- proc hex2dec {hexvalue} {
- set decvalue [format "%u" [expr 0x$hexvalue]]
- return $decvalue
- }

Convert decimal value to hexadecimal

- proc dec2hex { decvalue } {
- set hexvalue [format "%#010X" [expr $decvalue]]
- }

Print memory values – this keeps it top-of-mind for anyone running the script
return $hexvalue
}

proc Hex2IP { Hex } { Convert a hexadecimal value to it's equivalent IP address
  # first trim off leading "0x" if it's there
  if { $string length $Hex == 10 } { set Hex [string range $Hex 2 9] }
  binary scan [binary format H8 $Hex] c4 IPtmp
  foreach num $IPtmp {
    lappend IP [expr ($num + 0x100) % 0x100]
  }
  set IP [join $IP .]
  return $IP
}

proc isipvalid { IP } { Is the ip address valid?
  # only digits'n'dots
  regsub -all \[.0-9\] $IP {} scratchvar Are all the chars either numeric or periods?
  if { $scratchvar != "" } {
    return 0
  }
  # 4 octets means exactly 3 dots
  regsub -all \([0-9]\) $IP {} scratchvar Do we have exactly 3 periods?
  if { $scratchvar != "..." } {
    return 0
  }
  # is each octet betw 0 and 255?
  foreach b [split $IP .] {
    if { $string length $b == 0 } {
      return 0
    }
    set ob $b
    #parse out leading zeros
    scan $b %d b
    if { $b < 0 | $b > 255 } {
      return 0
    }
  }
  return 1 Final case, all is well
}

proc iscidrvalid { CIDR } { Is the network subnet mask (in CIDR notation) valid?
  # numeric check
  regsub -all \([0-9]\) $CIDR {} scratchvar Are all the chars either numeric or periods?
  if { $scratchvar != "" } {
    return 0
  }
  #convert to numeric, check values
  #because this is running on a router, mask <8 is not acceptable due to scan time.
  Values under 8 are too many IP's to scan on a router
  # mask of /31 or /32 is also not acceptable Values over 30 mess up the math, and are better specified as a range or discrete IP's anyway
  scan $CIDR %d CIDR
  if { $CIDR < 8 | $CIDR > 30 } {
    return 0
  }
proc ipCIDR { net } {
    global iplist
    set work1 [ split $net / ]
    set ip1 [ lindex $work1 0 ]
    if { ![isipvalid $ip1] } {
        puts stdout "Invalid IP address specified ==> $ip1"
        puts ""
        return 1
    }
    scan $net {%d.%d.%d.%d/%d} a b c d bits
    if { ![iscidrvalid $bits] } {
        puts stdout "Invalid Netmask address specified ==> /$bits"
        puts stdout "Because of platform considerations, subnet mask must be >=8 or <=30"
        puts ""
        return 1
    }
    set hexmask [expr {0xffffffff & (0xffffffff << (32-$bits))}]
    Get the broadcast ip address
    set bnet [ hex2dec [IPtoHex $ip1] ]
    set realnet [ expr $bnet & $hexmask ]
    The first ip is the network + 1
    set firstip [ expr $realnet+1 ]
    The first ip is the network + 1
    set bcast [ expr $bnet & ( $hexmask ^ 0xffffffff )]
    The last ip to scan is the broadcast – 1
    for { set j $firstip } { $j <= $lastip } { incr j} {
        Now, loop from first to last IP, and populate the IP list
        set work1 [ dec2hex $j ]
        lappend iplist [ Hex2IP $work1 ]
    }
    return 0
}

proc iprange { net } {
    Parse out a network, specified as with a range in octet 4
    global iplist
    set work1 [ split $net - ]
    set ip1 [ lindex $work1 0 ]
    Get the first ip
    set maxoct4 [ lindex $work1 1 ]
    Get the last ip
    if { ![isipvalid $ip1] } {
        Are both ip’s valid?
        puts stdout "Invalid IP address specified ==> $ip1"
        return 1
    }
    scan $ip1 {%d.%d.%d.%d} a b c d
    set ipmax $a.$b.$c.$maxoct4
    if { ![isipvalid $ipmax] } {
        puts stdout "Invalid IP address specified ==> $ipmax"
        return 1
    }
}
if { $d > $maxoct4 }  {  
    puts stdout "Invalid IP address range specified ==> $ip1-$maxoct4"  
    return 1  
}
for { set j $d; ($j <= $maxoct4) } { incr j; } {  
    lappend iplist $a.$b.$c.$j  
}
return 0

proc parsenet { networklist } {  
    Parse the complete IP list out from the cmd line string
    global iplist
    set netlist [split $networklist ,] Split out the commas
    foreach net $netlist {  
        if { [string first / $net] >0 } {  
            Is it a CIDR block?
            set retval [ipCIDR $net]  
        } elseif { [string first - $net] >0} {  
            Is it specified as a range?
            set retval [iprange $net]  
        } else {  
            if { ![isipvalid $net] } {  
                Is it a single ip address?
                puts stdout "Invalid IP address specified ==> $net"
                return 1
            } lappend iplist $net  
        }
    }
    return 0
}

proc pinger {ip timeout} {  
    Ping a host and tell me if it exists
    set pingretry 3  
    # returns a 1 if any icmp echo replies make it back, otherwise returns a 0
    if { ![regexp "(!)" [exec "ping $ip timeout $timeout repeat $pingretry"]]} {  
        return 1
    } else {  
        return 0
    }
}

proc scantcpconnect {host port} {  
    TCP Connect scan of a discrete ip address and port
    global timeout  
    global reason  
    set timeout1 [expr $timeout*1000]  
    Convert the timeout to milliseconds
    catch { socket $host $port } sock   
    Connect
    after $timeout1 Wait for the timeout
    if { [string first sock $sock] == 0} {  
        If the string “sock” is returned from the socket command, the port is open
        catch { close $sock }  
        return "open "
    } else {  
        return "closed"  
    }
}

proc udpscan { ip port } {  
    Attempt to see if a UDP port is open on a single ip address
    # timers should be global, logfile should NOT be global
    global timeouts  
    global waittime  
    global reason  
    ios_config "no logging buffer" Clear the buffered log (this is a config change)
# set up the list of interesting packets to look for
# (ie set up packet capture filter)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios_config &quot;access-list 111 permit udp any host $ip eq $port&quot;</td>
<td>Create the access list that identifies what we’re looking for</td>
</tr>
<tr>
<td>ios_config &quot;access-list 111 permit udp host $ip eq $port any&quot;</td>
<td></td>
</tr>
<tr>
<td>ios_config &quot;access-list 111 permit icmp host $ip any unreachable&quot;</td>
<td></td>
</tr>
<tr>
<td># now, watch for these packets (ie start your packet capture)</td>
<td></td>
</tr>
<tr>
<td>exec &quot;debug ip packet 111 det&quot;</td>
<td>Log occurrences of matches to our ACL to the log (ie – capture packets)</td>
</tr>
<tr>
<td># next, send test udp packets to trigger responses</td>
<td></td>
</tr>
<tr>
<td>ios_config &quot;ip sla monitor 111&quot; &quot;type udpEcho dest-ipaddr $ip dest-port $port control disable&quot; &quot;time $timeoutms&quot; &quot;freq 1&quot;</td>
<td>Now, let’s create an IP SLA to send some udp probe packets</td>
</tr>
<tr>
<td>ios_config &quot;ip sla mon schedule 111 life forever start now&quot;</td>
<td>Note that control packets are DISABLED</td>
</tr>
<tr>
<td>after $swaittime ; # wait - 2sec is generally enough for the log to</td>
<td>Schedule the IP SLA to run</td>
</tr>
<tr>
<td>catch up</td>
<td></td>
</tr>
<tr>
<td># now clean up config and debug changes</td>
<td></td>
</tr>
<tr>
<td>exec &quot;no debug ip pack 111 det&quot;</td>
<td>Clean up:</td>
</tr>
<tr>
<td>• Stop the packet capture</td>
<td></td>
</tr>
<tr>
<td>ios_config &quot;no access-list 111&quot;</td>
<td>• Clear the ACL</td>
</tr>
<tr>
<td>ios_config &quot;no ip sla monitor 111&quot;</td>
<td>• Erase the IP SLA</td>
</tr>
<tr>
<td>set startpos &quot;dst=$port&quot;</td>
<td></td>
</tr>
<tr>
<td>set logfile [ exec &quot;show log&quot; ]</td>
<td>Move the log into a variable list</td>
</tr>
<tr>
<td>set ipstart 0</td>
<td></td>
</tr>
<tr>
<td>set portunreach 0</td>
<td></td>
</tr>
<tr>
<td>set unreach 0</td>
<td></td>
</tr>
<tr>
<td># first, find the first occurrence of our target in the log</td>
<td></td>
</tr>
<tr>
<td>set ipstart [ string first $startpos $logfile ]</td>
<td>Where is the first occurrence of a sent packet?</td>
</tr>
<tr>
<td># now, look for icmp type 3, or icmp type 3 code 3, occurring after this ip value</td>
<td></td>
</tr>
<tr>
<td># (ie – make sure we’re not reading a previous status).</td>
<td>Look for ICMP Port unreachable replies returned after this packet</td>
</tr>
<tr>
<td>if { $ipstart &gt; 0 } {</td>
<td></td>
</tr>
<tr>
<td>Set unreach [ string last &quot;ICMP type=3&quot; $logfile ]</td>
<td>Find the last ICMP UNREACHABLE</td>
</tr>
<tr>
<td>Set portunreach [ string last &quot;ICMP type=3, code=3&quot; $logfile ]</td>
<td>Find the last ICMP PORT UNREACHABLE</td>
</tr>
<tr>
<td>set udpreturn [ string last &quot;UDP src=$port $logfile ]</td>
<td>Find the last UDP port from the target</td>
</tr>
<tr>
<td>set retcode &quot;open/filtered&quot; ; # set the case for no packets back at</td>
<td>The default case is no packets back – open/filtered</td>
</tr>
<tr>
<td>all</td>
<td></td>
</tr>
<tr>
<td>if { $unreach &gt; $ipstart } { set retcode &quot;filtered&quot; }</td>
<td>ICMP unreachable indicates filtered</td>
</tr>
<tr>
<td>if { $portunreach &gt; $ipstart } { set retcode &quot;closed&quot; }</td>
<td>Is it closed? (ICMP port unreachable) – overwrites &quot;filtered&quot; case above as it’s a more specific ICMP unreachable</td>
</tr>
<tr>
<td>if { $udpreturn &gt; $ipstart } { set retcode &quot;open&quot; }</td>
<td>Return packet indicates open port</td>
</tr>
<tr>
<td>} else { set retcode &quot;open/filtered&quot; } ; # this accounts for no packets back on empty logfile</td>
<td>This case is for an empty logfile (should never occur)</td>
</tr>
<tr>
<td>return $retcode</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

proc scanit [localportlist localnetworklist scantype pingit] {          |                                                                            |
| Generic scan a network list with a portlist, all scan types           |                                                                            |
| global timeout                                                       |                                                                            |
| global svctcp                                                         |                                                                            |
foreach host $localnetworklist { For each host in the list ....

# set existance default in case -P0 (no ping) is specified
set hostexist 1 If we're not pinging the hosts, tell me that they are all up.
if {$pingit == 1} { set hostexist [pinger $host $timeout] } If we're pinging, do it

if {$scantype == "P"} { Is it a ping scan?
    puts stdout "Host $host is up"
} else { puts "Host $host is down" }

if {$hostexist == 1} { If so, simply print the results and go on to the next
    puts stdout "Interesting ports on host $host"
    puts -nonewline stdout "PORT     STATE      SERVICE"
    if {$reasoncode == 1} {puts -nonewline stdout "    REASON"}
    puts ""
    foreach port $localportlist { Print the port title line – REASON is only printed if requested
        if {$scantype == "T"} { Is it a TCP scan?
            set state [ scantcpconnect $host $port ] If so, proceed
            set proto "tcp"
            if {$port <= $svcarraymax} { Format the results
                set service $svctcp($port)
            }
        } elseif {$scantype == "U" } { Is it a UDP scan?
            set state [udpscan $host $port ] If so, proceed
            set proto "udp"
            if {$port <= $svcarraymax} { Format the results
                set service $svcudp($port)
            }
        } elseif {$scantype == "L" } { Is it a list scan?
            set state "unscanned"
            set state "unscanned"
            if {$port <= $svcarraymax} { Format the results
                set service $svctcp($port)
            }
        }
        puts -nonewline stdout "$port/$proto   $state     $service"
        if {$reasoncode == 1} { puts -nonewline stdout "    $reason"
        puts stdout ""
    }
    On to the next port
} else { puts stdout "Host $host is unavailable" } Host is not up
    puts stdout ""
} On to the next ip
}
return
}

proc parseports { ports } { Parse the ports out from the command line string
    global portlist
    set localportlist [split $ports ,]
    global svcudp
    global svcarraymax
    global reason
    global reasoncode
foreach port $localportlist {
    if {[string first - $port] > 0} {  
        Is this a port range?
        set localplist [split $port -]
        for {set lport [lindex $localplist 0]} {$lport <= [lindex $localplist 1]} {incr lport} {
            Loop through
            if {$lport > 0 && $lport <65535 } {
                Ensure that we are >0 and < 64K
                lappend portlist $lport
            } else {
                puts stdout "Invalid port value ==> $lport"  
                Invalid port error
                return 1
            }
        }
    } else {
        This is a single port - again, check
        lappend portlist $port
    } else {
        if {$port >0 && $port <65535 } {
            This is a single port - again, check
            lappend portlist $port
        } else {
            puts stdout "Invalid port value ==> $port"  
            return 1
        }
    }
} return 0

proc getservices { loadpath } {  
Get the services file (we have names for the same services as NMAP does
    global svctcp
    global svcudp
    global svcarraymax
    for {set i 1} {$i < $svcarraymax} {incr i} {  
        Populate the array with dots first (undefined ports)
        set svctcp($i) "."
        set svcudp($i) "."
    }
    set svcfile "/services.list"
    if [catch {open $loadpath$svcfile r} fileId] {  
        Now, get the services file from the same location we loaded IOSmap from.
        puts stderr "Cannot open services file" Error in file read
    } else {
        set services [read $fileId]
        Read the file into a temp list
        close $fileId
    }
    foreach record $services {  
        For each line in the file
        set localrec [split $record ,]
        set localproto [lindex $localrec 0]
        set localport [lindex $localrec 1]
        set localsvc [lindex $localrec 2]
        switch $localproto {
            tcp  
                set svctcp($localport) $localsvc  
            Populate the tcp and udp service description arrays
            udp  
                set svcudp($localport) $localsvc
            }
        return
    }
Main Script Execution Starts Here

```bash
#process cmd line arguments
foreach arg $argv {
    switch -glob -- $arg {
        -sU {set scantype U} Populate the appropriate variables
        -sT {set scantype T}
        -sP {set scantype P ; set ports 1}
        -sL {set scantype L ; set ports 1 ; set pingit 0}
        -P0 {set pingit 0}
        -p* {set ports $arg}
        -h { set scantype "H" }
        default {set network $arg} If there is no switch, assume that it's a network value
    }
}

if {$scantype != "P"} { Is it NOT a Ping sweep?
    set loadpath [ setloadpath $fullpath] (Services file is not required for pings)
    #populate the arrays defining the tcp and udp service descriptions
    #depends on a data file in a hard-coded directory
    getservices $loadpath
}

# dump out intro line
puts stdout "\n\n"
puts stdout [clock format [clock seconds] -format {Starting IOSmap 0.9 (http://www.defaultroute.ca ) at %Y-%m-%d %H:%M %Z}]
puts ""

if {$scantype != "H" } { Is it NOT a request for syntax help?
    # trim -*p out of ports arg, parse out the ports to a list of discrete values
    set ports [string trimleft $ports -p] Pull the -*p off the port list
    set ok1 [parseports $ports] Now, get the port list

    # parse network values out to a discrete list of ip addresses
    set ok2 [parsetnet $network] Get the network list

    set ok [expr $ok1+$ok2 ]

    if { $ok == 0 } { Are the port list and network list both ok?
        set retcode [ memcalc $scantype ] Calculate the memory utilization
        if {$retcode == 0 } { Is the memory situation ok?
            # scan the list of ports and ip's as specified
            scanit $portlist $iplist $scantype $pingit If so, scan the ports
        } else {
            syntaxhelp Print the syntax help text
        }
    } else {
        syntaxhelp Print the syntax help text
    }
}
```

# Upcoming Training

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS London October 2020</td>
<td>, United Kingdom</td>
<td>Oct 12, 2020 - Oct 17, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS October Singapore 2020 - Live Online</td>
<td>Singapore, Singapore</td>
<td>Oct 12, 2020 - Oct 24, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Dallas Fall 2020</td>
<td>Dallas, TX</td>
<td>Oct 19, 2020 - Oct 24, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London November 2020</td>
<td>, United Kingdom</td>
<td>Nov 02, 2020 - Nov 07, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS San Diego Fall 2020</td>
<td>San Diego, CA</td>
<td>Nov 16, 2020 - Nov 21, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Frankfurt November 2020</td>
<td>, Germany</td>
<td>Nov 30, 2020 - Dec 05, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Cyber Defense Initiative 2020</td>
<td>Washington, DC</td>
<td>Dec 14, 2020 - Dec 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Cyber Security Central Jan 2021</td>
<td>,</td>
<td>Jan 18, 2021 - Jan 23, 2021</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Online</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s Only</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>