**Advanced Host Detection**

Techniques To Validate Host-Connectivity

whitepaper by dethy
dethy@synnergy.net

Abstract

Security Engineers spend a tireless amount of effort to block and filter packet anomalies in an internetwork connected environment. Advanced host mapping bypasses many forms of intrusion detection systems, filters, and routers, essentially enabling an attacker to map and discover previously unknown firewalled hosts.

## Introduction

This paper will attempt to describe techniques used to discover heavily filtered and firewalled hosts, that will not answer to standard PING responses. It is assumed that the reader has a firm knowledge of the major internet protocols (TCP,IP,UDP,ICMP). Most other protocols will not be discussed but techniques described here can be applied to many protocols.

## Host Detection Methods

It is becoming increasingly apparent the amount of firewalled and filtered hosts connected to the internet nowadays. Misconfigured and intrinsically firewalled hosts often block packet responses and replies that determine their (inter)network connectivity. A prime example of this scenario is the standard PING (packet internet groper) utility. PING issues an ICMP type 3 (echo request) response to an arbitrary host to test for it's online connectivity. However, since a growing number of these servers block many forms of ICMP code types, a reply will often be blocked, dropped and thus undelivered. Unfortunately, a client may then assume the network or host is down or inconveniently firewalled.

Exactly how can one knowingly detect the online presence of a host ?

Understanding avenues which can circumvent certain levels of firewall rulesets, will ultimately allow a client to determine whether a host is network connected and/or behind a filtered environment. This technique is known as 'Host Detection.

Host detection is similar to scanning in several ways although host detection does not test for the absence of packets to ports or modifications pertaining to protocol headers, ie setting flagged packet replies, but rather tests any responsiveness signs of issued from the remote host. In this respect, host-detection is a form of PING scanning, that is detecting any form of response to signify the apparent connective state of a server.

This paper analyses two broad 'PING sweep' host detection techniques that can be used in a (inter)networked environment for advanced host mapping.

* eliciting valid protocol responses
* generating invalid server-side protocol responses

The first method includes eliciting valid responses from supported protocols on a host. Any valid request from a client issued to a server over

TCP/IP/UDP/ICMP that will assume a reply, in terms of an answered request, is confined into this category. Such methods include:

* UDP Echo
* TCP Echo
* UDP Closed Ports
* TCP ACK
* TCP SYN
* TCP SYN|ACK
* TCP FIN
* TCP NULL FLAGS
* TCP XMAS
* ICMP Echo Request (Type 8)
* ICMP Broadcast
* ICMP Router Solicitation (Type 10)
* ICMP Timestamp Request (Type 13)
* ICMP Information Request (Type 15)
* ICMP Address Mask Request (Type 17)

Opposing these RFC-compliant replies are the underlying methods to generate invalid responses from the target host in order to determine its hidden presence. Of course receiving a reply from any of these methods will allow us to knowingly detect whether a host is online and/or firewalled. These methods include:

* Timedout Packet Fragmentation
* Invalid IP Header Length
* Invalid IP Field Values

## Eliciting Valid Protocol Requests

The first definitive category of host detecting takes place in the form of eliciting valid protocol queries. Several such methods are included using valid packet requests.

* Echo port method
* UDP method
* TCP FLAG method
* ICMP request method

All of the above categories are possible methods that allow any arbitrary client to request a returned packet reply in order to determine it's interconnectivity. As such, the packets returned and transmitted are valid protocol responses, and thus is differentiated from generating invalided responses since each request correctly uses TCP/IP/UDP/ICMP protocols without mangling any of the available fields.

## ECHO Port Method

This old-fashioned and outdated technique used to determine a host responsiveness at a very basic level can be still used on poorly/misconfigured UNIX hosts. Most often a security conscious administrator will block traffic to port 7 TCP/UDP or disable this service which
runs from inetd.

### TCP/Echo Port

This simplistic method uses a standard three-way TCP handshake that aims to establish a connection to the echo port (7/tcp). If a connect()'ion is established the host is then assumed as being online and thus the host detection sequence has taken place at this very basic level. As such, since the three-way handshake along with the potential of the echo port being open and even firewalled, makes this method highly restricted and problematic.

Although most UNIX/Linux distributions have made the echo port disabled by default it is still in use on many systems. The diagnostical purposes that this server was initially set out to achieve has become far out-weighted by the security implication that it opens up as a result of the increasing traffic that it may generate on a connecting client (which in turn diminishes it's own bandwidth and system processor performance).

Since three-way handshaking begins with an initial SYN flag packet and receives a SYN|ACK in reply, a client does not need to continue with the handshaking paradigm in order to determine the hosts responsivesness. Not only would it log but it has the highest chance of being noticed and consecutively blocked by the arbitary host. As such misconfigured or simplistic configuration of a firewall would allow packets with the SYN flag set to pass through. Noteably, this is why the TCP echo port method should only be used as a last resort - and even then it's not a wise idea, unless your aim is to inevitably trigger most forms of intrusion detection systems (IDS) and alarm prudent systems administrators.

An example of using this method in a networked environment could be accomplished through telnet.

```
dethy@dev:~ $ telnet XXX.XXX.XXX.XXX 7
Trying XXX.XXX.XXX.XXX...
Connected to XXX.XXX.XXX.XXX
Escape character is '^]'.
Hello.
Hello.
```

As shown above, the remote host replies to our initial 'Hello.' with its own 'Hello.', obviously the server is responsive. Creating a scanner for such a method wouldn't necessarily need to send any garbled data to the port, an established connection is all that is required.

Note: Other methods to determine whether this service is running on a remote host which avoid the TCP three-way handshake could alternatively be used for such purposes as defeating packet loggers.
Check http://www.synnergy.net/Archives/Papers/dethy/portscan.pdf for a further discussion of advanced port scan techniques.

## UDP/Echo Port

Similarly to the TCP echo port method, the UDP port 7 will answer a clients datagram with it's own UDP datagram. Since the packet block initially sent is replied with an answer from a remote host, we know the host is alive.

Using hping (available from http://www.kyuzz.org/antirez/hping2.html) as our packet generator we send the following:

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -2 -p 7
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): udp mode set, 28 headers + 0
data bytes
50 bytes from XXX.XXX.XXX.XXX: seq=0 ttl=64 id=1255 rtt=0.9 ms
```

## UDP Method

This next technique involves the User Datagram Protocol and it's known responsiveness to closed ports. This clandestine packet reply is taken from the known UDP port scan method. The logic involved with this is by sending a UDP datagram to a closed, NON- LISTENING port, the arbitrary host should respond with an ICMP_PORT_UNREACH error message. Since this host returns such a response, we are then able to determine and indicate it's connectivity - and thus is assumed alive.

The cycle for this method is as follows:

* client -> UDP (to closed port)
* server -> ICMP_PORT_UNREACH

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -2 -p 65
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): udp mode set, 28 headers + 0
data bytes
ICMP Port Unreachable from XXX.XXX.XXX.XXX  (XXX.XXX.XXX.XXX)
```

Predicting a closed port is fairly simple. Try choosing a high port (greater than 1024 and less than 65536). Of course if a reply to a UDP datagram is not sent back to the client, evidence shows that this packet would have been dropped by the kernel since the destination port would have been open or a filtering system has blocked the packet. Naturally, if this scenario occurs choose another port to test for responsiveness (one that is closed).

Caution is to be taken with UDP packets, however. Since UDP are often dropped during transmission, and/or blocked by firewalls, a replied ICMP_PORT_UNREACH may not even arrive to the client at all. In this instance retransmission should take place for added certainty.

## TCP FLAG Methods

Streaming various flagged packets over a network is perhaps the most effective method to determine the connectivity of a host. Since these packets are elusive in terms of transmission and presents itself as normal day to day traffic, they are rather difficult to differentiate between intrusive information gathering packets and harmless inbound traffic. All that is required for this host detection technique to be successful is a single flagged packet unlike the aforementioned TCP/UDP echo port method, involving three-way handshaking.

## TCP SYN Approach

This SYN flag method is a highly successul PING sweep implementation. Since a response is replied to any SYN packet on a closed or open port, it allows a client to be certain in detecting the presence of a host machine.

Manipulating the packet header to contain the SYN flag and transmitting this to an open port will return a packet with the SYN|ACK bits set. If no packet is returned at all, then a client may assume the host is firewalled, or or the port filtered.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -S -p 23
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): S set, 40 headers + 0 data bytes
```

```
50 bytes from 192.168.1.1: flags=SA seq=0 ttl=64 id=1252 win=32696 rtt=0.9 ms
```

As displayed the SA (SYN|ACK) is set in the returned packet. Being a blind client, one that does not completely know whether a port is open or closed does not matter in this host detection method. Since both open/closed ports will respond to the SYN packet, we do not particularly need to send the initial packet with knowledge of the state of the port whether open or closed.

An example sending a SYN to a closed port is shown below.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -S -p 2
eth0 default routing interface selected (according to /proc)
HPING atlanta (eth0 XXX.XXX.XXX.XXX): S set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=RA seq=0 ttl=255 id=1254 win=0 rtt=0.7 ms
```

The RA (RST|ACK) flags returned in this packet are indicative of a closed ports response. Since we receive a returned packet, we know the host is alive.

TCP ACK Approach

This method is arguably the most effective approach for PING scanning on a remote host. Flagging the ACK bit in the TCP header and transmitting the packet to an open or closed port should return a packet with the reset(RST) bit flagged. Evidently this method is disguised as normal traffic but also is flexible in that an open/closed port will not deter the end result. The state of the port for this method is not restrictive as stated, lucratively enabling a client to query a target on any given port (1-65536) and almost guaranteed a response.

Of course, hosts may disregard these packets with an effective rule base for routers, and firewalls. It is favorably plausible to determine whether a host is protected by some sort of filtering system through using a combination of the techniques this paper describes. Example, if a machine is found to be blocking TCP echo port connection, and no returned packets are replied at all, then using the TCP ACK approach directed to the echo port 7 will help enable the client to predict and spot rulesets by using the TCP ACK method as a diagnostical assumption. The server may reply with the RST bit, meaning:

* TCP echo port 7 is filtered by some arbitrary ruleset
* Packets with the SYN flag enabled are blocked to port 7
* TCP ACK packets are allowed through

All of the above are ostensibly obvious, but perhaps the assumption made about the SYN but may seem incorrect. However, through a process of elimination we know that the echo port is filtered, and we know that to establish a connection on this port we need to use the three-way handshaking negotiation, which involves the following responses:

SYN -> SYN|ACK -> ACK

Now we also know that the ACK packets returned the client with the RST bit in response. Eliminating the ACK flag from the above equation, we end up with SYN and SYN|ACK. Since the SYN flag is transmitted first from the client to the target and a SYN|ACK response is not being returned from the target we are able to cancel the SYN|ACK bit (since the SYN packet was not actually received, by means of some firewall blocking it’s transmission). Therefore, by some ruleset or firewall, packets matching the SYN flag are dropped on the receivers end. This is a technique known as firewalking, that is analysing the types of packets that are and aren’t allowed through in order to map the types of rulesets an arbitrary host has implemented (and those it has not).

Using HPING to issue an ACK packet to a closed and then an open port outputs the following:

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -A -c 1 -p 2
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): A set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=R seq=0 ttl=255 id=1048 win=0 rtt=0.5 ms
```

The -p argument denotes the port in which to send the packet. In this instance packet transmission was directed to port 2 (a NON-LISTENING port).

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -A -c 1 -p 23
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): A set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=R seq=0 ttl=255 id=1052 win=0 rtt=0.5 ms
```

where port 23 was an open, LISTENING port.

As is shown, both replies from the XXX.XXX.XXX.XXX host responded with the RST flag on open and closed ports. Thus we have verified this host exists(online).

TCP SYN|ACK Approach

This method is not the most flexible in terms of compatibility. BSD networking code does not send any flagged packet back to a SYN|ACK query packet, hence is architecture dependant. However, Linux/Windows detection (and others) can be obtained successfully with this technique.

A SYN|ACK packet is initially sent to an arbitrary port (open/closed state does not matter). The returned packet should be set with the RST bit in reply. Since the state of the port plays no role in this scenario, any random port could be used as the testing port.

The example shown below is a SYN|ACK packet issued to a Win95 machine with a non-listening port (23). The result is an RST flagged packet.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -S -A -p 23
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): SA set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=R seq=0 ttl=128 id=31029 win=0 rtt=0.5 ms
```

Likewise, the same packet is issued to a Linux machine except with a listening port on 23. The result is once again, an RST flagged packet.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -S -A -p 23
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): SA set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=R seq=0 ttl=255 id=1258 win=0 rtt=0.5 ms
```

TCP FIN Approach

Much more clandestine in approach is the TCP FIN host scan technique. Issuing a packet with this flag set to a closed port will return an RST|ACK packet from the remote host. Alternatively an open port will discard the packet and hence is useless to us as host detection extraodinaires.

Locating a closed port is clearly basic, take a random guess at a port above the reserved services(1-1024) where the abundance on the unserviced ports remain.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -F -p 2
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): F set, 40 headers + 0 data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=RA seq=0 ttl=255 id=1260 win=0 rtt=0.5 ms
```

The outbound packet was sent to a closed port with the RST|ACK bit replied. Since we our queried packet was replied by the server, we know the host is alive and well.

Alternatively the query packet does not invoke a reply from the remote host any of the below concepts may tell us why.

* inbound FIN packets blocked by firewall/router/ACLS
* inbound traffic to that port is filtered
* the queried port was open (try another port)
* host is down (unconnected from the (inter)network)

## TCP NULL Approach

This method involves unsetting all the flags in the TCP header and sending the packet to a closed, NON-LISTENING port. The reply should be a packet with the RST|ACK bits set. An open port will not respond to this packet (discarded), so once more choose a port that is known to have no services running by default.

An example of this closed port state along with no flags set is displayed below.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -p 2
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): NO FLAGS are set, 40 headers + 0
data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=RA seq=0 ttl=255 id=1267 win=0 rtt=0.5 ms
```

Defining a port to be used as the testing port for this host detection is a relatively easy choice. The results for this scan can often go undelivered to the client since ACL’s and rulesets particularly check for unflagged packet queries. This method therefore, will not be as effective as the aforementioned ACK and SYN techniques, but of course is a useful method if the host does not answer to standard ICMP type 8 echo requests.

## TCP XMAS Approach

Similarly to the NULL flag header host detection method, XMAS scanning tests a closed ports response to a packet that has enabled all bits of the TCP header flags: SYN, ACK, FIN, RST, URG, PSH (the two reserved bits do not modify the outcome). This method is based on the UNIX/Linux/BSD TCP/IP stack implementation and will not always successfully work against Windows operating systems.

```
dethy@dev:~ # hping XXX.XXX.XXX.XXX -c 1 -p 2 -F -S -R -P -A -U -X -Y

eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): RSAFPUXY set, 40 headers + 0
data bytes
50 bytes from XXX.XXX.XXX.XXX: flags=RA seq=0 ttl=255 id=1380 win=0 rtt=0.6 ms
```

The RST|ACK bits are indicative that the host received our reply and has confirmed it with its own transmitted packet. Therefore the client interprets the arbitrary host as being alive and network connected.

Since open ports do not respond to this malformed packet request, using an open port for host detection is trivial.

## Comments

From the above information, circumstantial evidence suggests that host detection of some arbitrary host is easily identifible if that host is running a UNIX/Linux/BSD derivative since these operating systems answer many malformed packet requests. Contrastedly, Windows based operating systems have a tendancy to drop many anomalistic traffic, which ultimately prevents these hosts from being successfully detected (but not transparent to many of the scans detailed above) in a (inter)networked environment.

## ICMP Methods

The Internet Control Message Protocol(ICMP) is used for reporting errors in datagram processing, and is an integral part of IP. ICMP has not especially been well-researched as a form of host detection until recently a whitepaper written by Ofir Akfin describes ways ICMP can be used in a number of scenarious, including fingerprinting and inverse mapping.

With information security and it’s importance on the increase, system analysts are implementening ACL’s and an effective rulebase to block all forms of ICMP. Although not all forms are considered lethal (smurf broadcasts, excessive unreachable error replies) many forms of ICMP aid server communication in a networked environment (timestamping for example).

Host detection disregards the type of ICMP that is filtered and look for any signs of life elicited through some arbitrary ICMP type datagram. Today, most hosts have some form of filtering against ICMP type 8 (echo request) but have left other types, all the better for host detection.

## ICMP Echo Request (Type 8)

PING? PONG! At last we reach the standard and mandatory method used for host - detection. The ’PING’ network diagnostic utility elicits ICMP echo_request datagrams to analyse network connectivity. An echo_reply Type 0 ICMP datagram will be returned if the host is active(online).

Since this method was designed to be the standard method for host detection recognition; firewalls, routers, ACLs have designed their rulesets around this fact and have consequently blocked all forms of ICMP Type 8 inbound network traffic. This gives reasons to all the other techniques described above which evade standard echo responses (and are just as successful).

Focussing more directly at the ICMP Type 8 packet reveals the following:

```
    0                   1                   2                   3
```

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Type      |     Code      |          Checksum             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |           Identifier          |        Sequence Number        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             Data                              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A generic response is as follows:

```
dethy@dev:~ # hping -1 XXX.XXX.XXX.XXX -c 1 -C 8
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.XXX (eth0 XXX.XXX.XXX.XXX): icmp mode set, 28 headers + 0
data bytes
50 bytes from XXX.XXX.XXX.XXX: icmp_seq=0 ttl=255 id=1273 rtt=0.4 ms
```

As is displayed 50 bytes of the ICMP echo_reply were returned from the target host. Similarly, as is the case with other methods since we received an answered packet the host is assumed alive.

ICMP Broadcast

Broadcasting is a way of transmitting packets to all connected hosts of a network by sending an echo request(Type 8) to the network or broadcast address. The results will be a magnification of the first initiated packet with each networked host sending their own reply back to the instigating client.

In fact, ICMP Broadcasting is an extremely useful method to map an arbitrary network’s interconnected computers. Since each echo query is answered, it allows simple host detection for a prober to discover an entire network. However, there is a drawback. By default Windows computers (except NT 4 with Service Pack < 4) do not answer to ICMP Type 8 echo request packets directed to the broadcast or network address but instead silently discards any such packets. Once again it becomes apparent that Windows boxes can be rather elusive in terms of remote host detection.

Below is an example of an ICMP echo request packet sent to the network address of some server.
Note: The XXX.XXX.XXX.4 IP address did not return any reply since
it was a Windows95 box.

```
dethy@dev:~ # hping -1 XXX.XXX.XXX.0 -c 2
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.0 (eth0 XXX.XXX.XXX.0): icmp mode set, 28 headers + 0
data bytes
28 bytes from XXX.XXX.XXX.3: icmp_seq=0 ttl=255 id=13013 rtt=0.4 ms
50 bytes from XXX.XXX.XXX.1: icmp_seq=0 ttl=255 id=426 rtt=0.6 ms
50 bytes from XXX.XXX.XXX.2: icmp_seq=0 ttl=255 id=15319 rtt=0.8 ms

--- XXX.XXX.XXX.0 hping statistic ---
1 packets tramitted, 3 packets received, -100% packet loss
round-trip min/avg/max = 0.4/0.6/0.8 ms
```

It is noticed that the single transmitted packet received three replies when directed to the network address.

Alternatively, a packet sent to the broadcast address will likewise produce three answered reply datagrams.

```
dethy@dev:~ # hping -1 XXX.XXX.XXX.255 -c 2
eth0 default routing interface selected (according to /proc)
HPING XXX.XXX.XXX.255 (eth0 XXX.XXX.XXX.255): icmp mode set, 28 headers + 0
data bytes
28 bytes from XXX.XXX.XXX.3: icmp_seq=0 ttl=255 id=13098 rtt=0.4 ms
50 bytes from XXX.XXX.XXX.1: icmp_seq=0 ttl=255 id=730 rtt=0.7 ms
50 bytes from XXX.XXX.XXX.2: icmp_seq=0 ttl=255 id=15327 rtt=0.8 ms

--- XXX.XXX.XXX.255 hping statistic ---
1 packets tramitted, 3 packets received, -100% packet loss
round-trip min/avg/max = 0.4/0.7/0.8 ms
```

Once again, this technique has shown a successful method using address broadcasting as a network host mapping mechansism.

ICMP Router Solicitation (Type 10)

The ICMP router discovery requests are called Router Solicitations. Each router periodically multicasts a Router Advertisement (ICMP Type 9) from each of its multicast interfaces, which in turn announces the IP address(es) of that interface.

This technique is useful for discovering a system acting as a Router. It is known that ICMP Router Solication is an optional message format on a standard host. However, it is mandatory for a router to have enabled the ICMP Router Solication implementation. Thus, if servers respond with an ICMP Type 9 in reply to an ICMP Type 10, one can be fairly certain that the server is a router or network device. Needless to say, a host that receives an ICMP Type 10 but is not configured to transmit these messages, can not send back a reply.

The packet format for a router discovery messages looks like the following:

```
 0                   1                   2                   3
 0   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(Unfortunately hping has not implemented ICMP Type 10,13,15,17, message formats. Instead icmpush (available from http://hispahack.ccc.de) has been alternatively selected as the packet generating/analysing utility).

```
dethy@dev:~ # ./icmpush -vv -rts XXX.XXX.XXX.XXX
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> ICMP total size = 20 bytes
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 40 bytes
ICMP Router Solicitation packet sent to XXX.XXX.XXX.XXX (XXX.XXX.XXX.XXX)

Receiving ICMP replies ...
XXX.XXX.XXX.XXX -> Router Advertisement (XXX.XXX.XXX.XXX)
./icmpush: Program finished OK
```

Another successful hit! We know we have found a Router on this network. Perhaps the Router is filtering other ICMP types or perhaps blocking ports, but fortunately this paper has discussed alternative methods to bypass these forms of ACL.

ICMP Timestamp Request (Type 13)

The reply (ICMP Type 14) within a timestamp request is the initial request data additionally with the remote hosts timestamp. Obviously, timestamps requests are made in order to query a server for the current time.

Often cross platform compatibility issues lend a hand when requesting a timestamp reply. Windows95 and WindowsNT did not answer queries that were sent, UNIX/Linux/BSD replied with the correct data.

Taking a look at the ICMP packet itself reveals the following:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Originate Timestamp                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Receive Timestamp                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Transmit Timestamp                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The first example shown below transmits an ICMP timestamp request to a Linux server, the result was 03:50:32 encapsulated within the data field.

```
dethy@dev:~ # ./icmpush -vv -tstamp XXX.XXX.XXX.XXX
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> ICMP total size = 20 bytes
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 40 bytes
ICMP Timestamp Request packet sent to XXX.XXX.XXX.XXX (XXX.XXX.XXX.XXX)

Receiving ICMP replies ...
XXX.XXX.XXX.XXX -> Timestamp Reply transmited at 03:50:32
./icmpush: Program finished OK
```

The next example issued the same packet but to a Windows95 computer, no returned packet was captured.

```
dethy@dev:~ # ./icmpush -vv -tstamp XXX.XXX.XXX.XXX
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> ICMP total size = 20 bytes
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 40 bytes
ICMP Timestamp Request packet sent to XXX.XXX.XXX.XXX (XXX.XXX.XXX.XXX)
```

```
Receiving ICMP replies ...
./icmpush: Program finished OK
```

ICMP Timestamp request us a healthy method to use for host detection, particularly *NIX servers.

ICMP Information Request (Type 15)

This message is used to query a host to discover it's network address, however as the RFC states, ICMP Type 15 (Information Request) and ICMP Type 16 (information reply) are obsoleted, but that's not to say it's still not in use in the wild. :)

A cross section of this ICMP type reveals the following:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A packet with ICMP Type 15 was sent to a host which in turn answered the request, and thus gave indication of it's existence to some arbitrary host.

```
dethy@dev:~ # ./icmpush -vv -info XXX.XXX.XXX.XXX
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> ICMP total size = 8 bytes
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 28 bytes
ICMP Info Request packet sent to XXX.XXX.XXX.XXX (XXX.XXX.XXX.XXX)

Receiving ICMP replies ...
XXX.XXX.XXX.XXX -> Info Reply (XXX.XXX.XXX.XXX)
./icmpush: Program finished OK
```

Once again, I could not reproduce these results against a Windows 95/NT system, but several *NIX distribution replied successfully.

ICMP Address Mask Request (Type 17)

Address mask requests are generated to obtain the subnet mask address on the local network. The response to this initial query packet will be an ICMP Type 18 (Address Mask Reply), which should contain the subnet address.

A detailed look at the ICMP Address Mask reveals the following:

```
 0                   1                   2                   3
 0   2   4   6  8  10  12  14  16  18  20  22  24  26  28  30  31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |      Code     |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Subnet Address Mask                      |
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Funnily enough issuing this ICMP type to various Linux boxes did not return an ICMP Type 18 reply, however, Windows systems did.
The example below shows the result of the Address Mask Request packet initiated against a Windows box.

```
dethy@dev:~ # ./icmpush -vv -mask XXX.XXX.XXX.XXX
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> ICMP total size = 12 bytes
 -> Outgoing interface = XXX.XXX.XXX.XXX
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 32 bytes
ICMP Address Mask Request packet sent to XXX.XXX.XXX.XXX (XXX.XXX.XXX.XXX)

Receiving ICMP replies ...
XXX.XXX.XXX.XXX -> Address Mask Reply (255.255.255.0)
./icmpush: Program finished OK
```

So the subnet mask was received at our end. Yet another host detection method is feasible against Windows systems to use for host mapping in a cross platform network environment that blocks ICMP echo requests.

Comments

Some readers may be thinking why not elicit ICMP reply's to hosts and analyse their response to use for host detection. RFC 1122 meaningfully states the following:

```
An ICMP error message MUST NOT be sent as the result of receiving:
   *    an ICMP error message, or
   *    a datagram destined to an IP broadcast or IP multicast
        address, or
   *    a datagram sent as a link-layer broadcast, or
   *    a non-initial fragment, or
   *    a datagram whose source address does not define a single
        host -- e.g., a zero address, a loopback address, a
        broadcast address, a multicast address, or a Class E
        address.
```

The first point clearly states that a host will not issue a response to an ICMP reply datagram, so much for host detection using reply datagrams. Oh well time put your thinking caps on once more. :)

Generating Invalid Protocol Responses

This section describes methods using the Internet Protocol (IP) to divulge error messages in order to discovery arbitrary hosts. The corresponding encapsulated protocol (TCP/UDP/ICMP) has no effect on the results using this method when packaging the datagrams.

The foundation for analysing the connectivity of a host against this method emphasises the need for effective ACL's and outgoing packet filtering. The basis for this technique relies on generating invalid datagrams and detecting external responses that the malformed packet creates as a result of the abnormality.

## IP Header Approach

Creating anomalistic IP headers in transmission will help increase the chances of detecting a firewalled and filtered host. Many forms of intrusion detection systems and routers do drop packets that contain malformed headers such as invalid field values. The techniques below list such scenarios.

* Timedout Packet Fragmentation
* Invalid Header Length
* Invalid Field Values

## Timedout Packet Fragmentation

Another method used in advanced host detection is unsent packet fragmenting. It is firstly necessary to construct a packet with a fragmented offset and send to a host. Instead of assembling another fragmented datagram to complete the packet, the client will let the initial fragmented datagram timeout, leaving the server waiting for the next expected packet in the sequence. The effect of this is an elicited ICMP Type 11 Code 1 Time Exceeded Fragment Reassembly generated by the server.

Example:

```
dethy@dev:~ # hping -c 1 -x -y XXX.XXX.XXX.XXX
eth0 default routing interface selected (according to /proc)
HPING dev (eth0 XXX.XXX.XXX.XXX): NO FLAGS are set, 40 headers + 0 data bytes

--- dev hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Note: Although hping returns 100% packet loss, if does not check for the ICMP datagram the remote host generated.

tcpdump shows the following information:

```
20:41:09.309085 YYY.YYY.YYY.YYY > XXX.XXX.XXX.XXX: icmp: ip reassembly time
exceeded [tos 0xc0]  (ttl 255, id 3375)
```

Once again we have produced a response from a server using invalid protocol communication.

## Invalid Header Length

Specifying an invalid header length within an IP header will result in the remote host generating an ICMP Type 12 - Parameter Problem error message. The Code type of this within this ICMP datagram may be equal to either of the following:

0 - Pointer indicates the error
2 - Bad Length

A code equal to 0 will return the exact byte which caused the error encapsulated within the pointer field. Alternatively a code equal to 2 signifies the entire packet contains errors. In either case, the host on the receiving end of this packet solicits the ICMP Type 12 Code (0 | 2) in return to tell the sender that the packet has been discarded or dropped.

Below ISIC (IP Stack Integrity Checker) was used to assemble a packet with an incorrect IP header length of 66 bytes.

```
 dethy@dev:~ # ./isic -s YYY.YYY.YYY.YYY -d XXX.XXX.XXX.XXX -p 1 -V 0 -F 0 -I 66
-D
 Compiled against Libnet 1.0.1b
 Installing Signal Handlers.
 Seeding with 5099
 No Maximum traffic limiter
 Bad IP Version  = 0%   Odd IP Header Length   = 100%   Frag'd Pcnt   = 0%
 YYY.YYY.YYY.YYY -> XXX.XXX.XXX.XXX tos[137] id[0] ver[4] frag[0]

  Wrote 1 packets in 0.00s @ 5649.72 pkts/s
```

tcpdump trace revealed the following:

```
  21:39:03.755839 XXX.XXX.XXX.XXX > YYY.YYY.YYY.YYY: icmp: parameter problem -
  octet 20 [tos 0xd0]  (ttl 255, id 21508)
```

As was expected, a malformed header length forced an arbitrary response from the server. This method ultimately could be used to bypass many forms of ACL's and filtering systems if not correctly configured.

Invalid Field Values

On a more general level, specifying invalid values within any fields of the IP header will produce ICMP errors messages on the target host. Such a case is with the IP PROTO field, which has a total of 8 bits in length and hence has a possible total of 256 (2^8) combinations. The trick involved in this instance is by electing a protocol value that is not indicative of a legal protocol value on that host.

Fortunately a client is able to determine if a host does not support a protocol, as the server will generate an ICMP Type 3 Code 3 - Destination Unreachable Protocol Unreachable. If a response is not sent back, the client assumes that this protocol specified is supported on that host.

For the next example apsend (http://www.elxsi.de) was used to generate the packet.

```
 dethy@dev:~ # perl apsend -s YYY.YYY.YYY.YYY -d XXX.XXX.XXX.XXX -b 8 -p 8
 --protocol 0
 Packet: 1 from YYY.YYY.YYY.YYY(port: 8) to XXX.XXX.XXX.XXX(port: 8).
 Protocol: 0  Type of Service(ToS): 16  ID: 0
```

In the above example the datagram was sent with a protocol equal to 0, and thus should always return an ICMP error.

A tcpdump trace returned the following data:

```
 21:58:21.128201 YYY.YYY.YYY.YYY > XXX.XXX.XXX.XXX: icmp: dev.synnergy.net
 protocol 0 unreachable [tos 0xd0]  (ttl 255, id 24133)
```

As expected XXX.XXX.XXX.XXX was returned with an ICMP Type 3 Code 3 datagram, once more we know the host is alive, thus another successful host detection method.

## Final Note

Further malformed packets could be used to generate arbitrary responses on a host using invalid IP field values, this is left for the reader to analyse. Most of these methods can be applied to most protocols such as IGMP or ARP as a useful mechanism to detect firewalled hosts.

Implementing inbound and outbound traffic filters is a must for any network wishing to avoid many forms of remote host detection. A proper rulebase and effective ACL's should be thouroughly reviewed and tested as a standard means of security practice.

By now the reader should be readily equipped with enough knowledge to accurately interrogate such protocols to generate server responses as a means of advanced host detection.

## References


ICMP Scanning - by Ofir Afkin - www.sys-security.com
RFC 792 - Internet Control Message Protocol
RFC 1122 - Requirements for Internet hosts - communication laye