

Identifying ICMP Hackery Tools Used In The Wild Today

Ofir Arkin 2000-12-04

Identifying ICMP Hackery Tools Used In The Wild Today

by Ofir Arkin ofir@sys-security.com

last updated Dec. 4, 2000

Introduction

Several tools exist in the wild today that allow a malicious computer attacker to send crafted ICMP datagrams. Those datagrams can be used for various tasks: host detection, advanced host detection, Operating System Fingerprinting and more. This article will examine whether we can identify the different tools used for ICMP hackery that are available in the wild today. If we can identify the tool, we may be able to identify the underlying operating system or a number of operating systems that this tool might be running on top of. We will use the fact that some of these tools inherit some values from the underlying OS. This query will be done passively, without actively querying the malicious computer attacker's machine.

In this discussion, we are going to examine five utilities:

1. The 'ping' utility from the iputils package with Redhat LINUX based on Kernel 2.2.14.
2. HPING2 beta 54, written by antirez (<http://sourceforge.net/projects/hping2>).
3. Nemesis v1.1, written by obecian (<http://www.packetfactory.net/Projects/Nemesis>) (<http://www.packetninja.net/nemesis/>).
4. Icmpenum v1.1.1, written by Simple Nomad (<http://razor.bindview.com>).
5. SING v1.0, written by Alfredo Andres Omell (<http://www.sourceforge.net/projects/sing>).

The type of ICMP message examined will be an ICMP Echo request sent with all tools examined. All tools were installed and used under a Redhat LINUX machine based on Kernel 2.2.14.

ICMP Echo Request

RFC 7922 - Internet Control Message Protocol (<http://www.ietf.org/rfc/rfc0792.txt>) - defines the way of sending an ICMP Echo request. The sending side initializes the identifier (used to identify Echo requests aimed at different destination hosts) and sequence number (if multiple Echo Requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP Echo to the destination host. In the ICMP header, the code equals zero. The recipient should only change the type to ECHO Reply and return the datagram to the sender (and the checksum of the ICMP header).

Differentiating Between the Tools

ICMP Echo Requests Using the "Ping" Utility with Linux

The following is an ICMP Echo request sent with the ping utility from the iputils package with Redhat Linux based on Kernel 2.2.14:

```
[root@godfather sbin]# ping -c 2 y.y.y.y
PING y.y.y.y (y.y.y.y) from x.x.x.x : 56(84) bytes of data.
64 bytes from hostname (y.y.y.y): icmp_seq=0 ttl=255 time=0.1 ms
64 bytes from hostname (y.y.y.y): icmp_seq=1 ttl=255 time=0.1 ms

--- y.y.y.y ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
[root@godfather sbin]#
```

The snort trace:

```
11/01-23:09:51.398772 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:38
ID:1037 Seq:0 ECHO
9F 86 00 3A 85 15 06 00 08 09 0A 0B 0C 0D 0E 0F ...:.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567
```

```
11/01-23:09:51.398819 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:39
ID:1037 Seq:0 ECHO REPLY
9F 86 00 3A 85 15 06 00 08 09 0A 0B 0C 0D 0E 0F ...:.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567
```

The following ICMP Echo request sent:

```
11/01-23:09:52.391176 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:40
ID:1037 Seq:1 ECHO
A0 86 00 3A EB F7 05 00 08 09 0A 0B 0C 0D 0E 0F ...:.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567
```

```
11/01-23:09:52.391220 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:41
ID:1037 Seq:1 ECHO REPLY
A0 86 00 3A EB F7 05 00 08 09 0A 0B 0C 0D 0E 0F ...:.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567
```

A few important fingerprinting facts about the ping utility must be kept in mind. Ping (like the other ping utilities with Unix and Unix-like operating systems) uses an ICMP data field of 56 bytes. This makes a total IP datagram length of 84 bytes. Ping starts its ICMP sequence number at 0, and the gap between one sequence number to another is 1 hex. Ping uses the value of 64 as its default value for the IP TTL field with ICMP Echo requests (and

with any ICMP Query Message type).

Another point to keep in mind is Linux behavior with IP ID numbers. The trace was taken after my Linux machine was booted. The first IP ID number that was used was of a decimal double-digit number. The gap between one IP ID number to another is 1.

Ping's ICMP ID number is the process ID assigned to ping when running.

HPING2

HPING2 is a network tool able to send custom ICMP/UDP/TCP packets and to display target replies like ping does with ICMP replies. HPING2 handles fragmentation, arbitrary packet body and size and can be used in order to transfer files under supported protocols. HPING2 compiles under Linux, and *BSD.

In the next example, I have sent an ICMP Echo request with HPING2:

```
[root@godfather sbin]# hping2 -1 -c 2 y.y.y.y
default routing not present
HPING y.y.y.y (y.y.y.y): icmp mode set, 28 headers + 0 data bytes
28 bytes from y.y.y.y: icmp_seq=0 ttl=255 id=36 rtt=0.8 ms
28 bytes from y.y.y.y: icmp_seq=1 ttl=255 id=37 rtt=0.7 ms

--- y.y.y.y hping statistic ---
2 packets tramitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/0.8 ms
[root@godfather sbin]#
```

The snort trace:

```
By Martin Roesch (roesch@clark.net, www.snort.org)
11/01-23:08:48.338644 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:14546
ID:1032 Seq:0 ECHO

11/01-23:08:48.338691 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:36
ID:1032 Seq:0 ECHO REPLY
```

One notable fact is that there is no data carried with the ICMP Echo request produced with the default behavior of HPING2. By default, the Total length of HPING2 ICMP Echo generated datagrams will always be 28 bytes.

Like ping, HPING2 is using its process ID as the ICMP ID field value:

```
[root@godfather /root]# ps aux | grep hping
root 4826 0.0 0.4 1200 512 pts/1 S 19:57 0:00 [root@godfather /root]#

11/04-19:57:44.846703 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:57750
```

```
ID:4826   Seq:20   ECHO
```

In the sendicmp.c the program states: icmp->un.echo.id = getpid();

This was only the first ICMP Echo request sent. What does the second ICMP Echo request look like?

```
11/01-23:08:49.331187 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:19756
ID:1032   Seq:1   ECHO
```

```
11/01-23:08:49.331233 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:37
ID:1032   Seq:1   ECHO REPLY
```

```
23:08:49.331187    > x.x.x.x > y.y.y.y: icmp: echo request (ttl 64, id 19756)
                   4500 001c 4d2c 0000 4001 2fb3 xxxx xxxx
                   yyyy yyyy 0800 eefb 0804 0100
```

Comparing the first pair of ICMP Echo request and reply with the second pair will reveal that they are not exactly the same. We can differentiate between them by:

- the different IP Identification numbers that were used.
- the sequence number used. The gap used is 1 hex.
- the ICMP header checksum (changed because the sequence number was changed).

We can also conclude that the IP ID field value is not in sync with the underlying operating system. In the examples above, HPING2 used 14546, and 19756, while the operating system was using 36, and 37 (if you do a test against the loopback address of your machine you can see this happening). In fact, the IP ID field value is randomly computed.

The only real detail we can use is the fact that HPING2 has no data in its ICMP data portion.

Nemesis

Nemesis is a TCP/UDP/ICMP/ARP/DNS/RIP/IGMP/OSPF network packet injection suite. Nemesis compiles under Linux, *BSD, and Sun Solaris.

In the next example I have sent an ICMP Echo request with Nemesis. The options I have used are -vv for verbose mode, -i for ICMP type, and -c for ICMP code value:

```
[root@godfather /root]# nemesis-icmp -vv -i 8 -c 0 -S x.x.x.x -D y.y.y.y
```

```
ICMP Packet Injection -- The NEMESIS Project 1.1
(c) 1999, 2000 obecian
```

```
[IP]   x.x.x.x > y.y.y.y
[Type] ECHO REQUEST
```

```
[Sequence number] 0
[IP ID] 0
[IP TTL] 254
[IP TOS] 0x18
[IP Frag] 0x4000
```

Wrote 48 bytes

ICMP Packet Injected

The tcpdump trace:

```
00:27:16.153322 > x.x.x.x > y.y.y.y: icmp: echo request (DF) [tos 0x18] (ttl 254, id 29)
```

```
4518 0030 001d 4000 fe01 7e95 xxxx xxxx
yyyy yyyy 0800 f7ff 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
00:27:16.153375 > y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x18] (ttl 255, id 30)
```

```
4518 0030 001e 0000 ff01 bd94 yyyy yyyy
xxxx xxxx 0000 ffff 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

Information that may help us identify Nemesis includes:

- the TOS field value is set to a value of 0x18 hex (by default it should be set to 0x00);
- the DF bit is set;
- the IP TTL field value was set to 254 (The underlying OS uses 64 as its default IP TTL for ICMP Echo Requests);
- the ICMP Identifier field value equal zero (ICMP_ID=0);
- the ICMP Sequence Number field value equal zero (ICMP_Seq=0); the ICMP data portion equals 20 bytes (Linux, Unix and Unix-like operating systems use 56 bytes of data with ICMP Echo Requests. Microsoft Windows operating systems use 32 bytes). This means that, by default, the total length of Nemesis ICMP Echo datagrams will be by 48 bytes long.
- the 20 bytes of data with this ICMP Echo request are all equal to zero.

This is the second pair of ICMP Echo request and reply sent by Nemesis:

```
00:27:23.294060 > x.x.x.x > y.y.y.y: icmp: echo request (DF) [tos 0x18] (ttl 254, id 31)
```

```
4518 0030 001f 4000 fe01 7e93 xxxx xxxx
yyyy yyyy 0800 f7ff 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
00:27:23.294097 > y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x18] (ttl 255, id 32)
```

```
4518 0030 0020 0000 ff01 bd92 yyyy yyyy
xxxx xxxx 0000 ffff 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

The second pair of ICMP Echo request and reply is identical to the first pair, except for the IP Identification number (which is used from the underlying operating system).

The checksum for the ICMP header is the same with both of the pairs, since the ICMP sequence numbers and the ICMP identification numbers used are the same. Since Nemesis will produce the same default ICMP Echo request each time we can track the same patterns.

Obecian is doing a major re-write of the tool, so in v2.0 there are going to be lots of surprises.

icmpenum

icmpenum is a proof of concept tool that was written by Simple Nomad3. The tool is able of sending ICMP Echo requests, ICMP Timestamp Requests, and ICMP Information requests.

```
[root@godfather /root]# ./icmpenum -i 1 IP_Address
```

```
11/04-19:40:22.256600 x.x.x.x -> y.y.y.y
ICMP TTL:255 TOS:0x0 ID:666
ID:39426   Seq:0   ECHO
```

```
11/04-19:40:22.256662 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:18
ID:39426   Seq:0   ECHO REPLY
```

```
19:40:22.256600 > x.x.x.x > y.y.y.y: icmp: echo request (ttl 255, id 666)
                4500 001c 029a 0000 ff01 bb44 xxxx xxxx
                yyyy yyyy 0800 f565 029a 0000
```

From the first ICMP Echo request we can conclude that the packet size icmpenum produce will be always 28 byte, without any data portion.

Sending another ICMP Echo request with icmpenum:

```
11/04-19:40:43.826947 x.x.x.x -> y.y.y.y
ICMP TTL:255 TOS:0x0 ID:666
ID:39426   Seq:0   ECHO
```

```
11/04-19:40:43.826992 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:19
ID:39426   Seq:0   ECHO REPLY
```

The IP ID will always be equal to the value of 029a hex / 666 decimal. The value of the ICMP ID will be always set to 029a hex. The ICMP sequence number will be always set to 0. The IP TTL with icmpenum is set to 255. These unique patterns allow us to identify icmpenum quite easily.

SING (Send ICMP Nasty Garbage)

SING is the "Swiss army knife" of ICMP. With SING you can send nearly everything your mind is thinking of regarding the ICMP protocol. SING compiles under Linux, *BSD, and Sun Solaris. In the following example I have sent an ICMP Echo request with SING:

```
[root@godfather /root]# sing -c 4 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=255 TOS=0 time=0.169 ms
16 bytes from y.y.y.y: seq=1 ttl=255 TOS=0 time=0.155 ms
16 bytes from y.y.y.y: seq=2 ttl=255 TOS=0 time=0.136 ms
16 bytes from y.y.y.y: seq=3 ttl=255 TOS=0 time=0.136 ms

--- y.y.y.y sing statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.136/0.149/0.169 ms
[root@godfather /root]#
```

```
00:24:59.011868 > x.x.x.x > y.y.y.y: icmp: echo request (ttl 255, id 13170)
      4500 0024 3372 0000 ff01 8a64 xxxx xxxx
      yyyy yyyy 0800 7cad 3c04 0000 2be6 e039
      332e 0000

00:24:59.011908 > y.y.y.y > x.x.x.x: icmp: echo reply (ttl 255, id 25)
      4500 0024 0019 0000 ff01 bdbd yyyy yyyy
      xxxx xxxx 0000 84ad 3c04 0000 2be6 e039
      332e 0000
```

The following information that may help us to identify SING:

- IP TTL used with the request is 255 (Underlying OS uses 64 as its default IP TTL field value with ICMP Echo requests);
- ICMP Identifier used 3c04 Hex, equal to 1084. The ICMP ID is actually the process ID number assigned to SING when running (like the ping utility);
- Only 8 bytes of ICMP data portion were used. They are used for the RTT calculation. This means that, by default, the total length of SING ICMP Echo datagrams will be 36 bytes.

This is the tcpdump trace of the second pair of ICMP Echo request & reply:

```
00:25:00.004593 > x.x.x.x > y.y.y.y: icmp: echo request (ttl 255, id 13170)
      4500 0024 3372 0000 ff01 8a64 xxxx xxxx
      yyyy yyyy 0800 d6c9 3c04 0100 2ce6 e039
      d711 0000

00:25:00.004630 > y.y.y.y > x.x.x.x: icmp: echo reply (ttl 255, id 26)
      4500 0024 001a 0000 ff01 bdbc yyyy yyyy
      xxxx xxxx 0000 dec9 3c04 0100 2ce6 e039
      d711 0000
```

From the second ICMP Echo Request & Reply pair we can conclude that:

- the IP ID Number used is the same;
- the sequence numbers used are not the same (Gap of 1 hex), therefore the ICMP Header checksum is not the same either.

Parameters we can use in order to identify SING are:

- IP ID will always be set to 3372 hex / 13170 decimal (under Linux & *BSD);
- ICMP data portion will be of 8 bytes in length (used for RTT calculation).

Concluding the Underlying OS

An Example with SING

If you bother to look deep into the sources, then you might reveal facts that will help you identify the underlying operating system that the tool was running on top of.

I will take SING for example. With Linux and *BSD operating systems SING is using a constant number for the IP ID field value. The value is 0x3372 hex (13170 decimal). Except for Sun Solaris operating systems, because specifying the IP ID is not allowed.

Since we can identify SING, looking at the IP ID field number and seeing a value different than 13170 decimal / 0x3372 hex would tell us that this particular SING tool runs on a Sun Solaris-based machine.

Another interesting piece of information we can use is the fact that Sun Solaris machines set their DF bit with ICMP Query Message types by default. SING is no exception. This gives us another extra verification for SING running under a Sun Solaris Machine.

A thought: the ICMP ID field value is the process ID assigned to SING by the underlying operating system. If we can calculate the range of numbers that will be assigned as a process ID to SING under the various operating systems, we might be able to conclude the underlying OS. For example - with Linux after boot, the ICMP ID was set to a value of 3c04 Hex, equal to 1084 decimal. With FreeBSD, the ICMP ID of SING was set to 1b03 hex / 795 decimal. With Sun Solaris, this value was set to 7330 hex / 12403 decimal. This is more complicated than it looks like, and should be further investigated to see if it can be trusted.

We can use all of this information for differentiating between the various platforms that SING might run under.

Other ICMP Query Message Types

We have examined only the ICMP Echo requests. If we were examining other ICMP Query message types, other sources for differentiation between the tools would be introduced. Some would be introduced because the only ICMP query message type, which is shared between all OSs, is ICMP Echo Request (with the ping utility).

Various Options with the Various Tools

Sometimes you will look at a trace of an ICMP datagram and be able to conclude that one of its fields was obviously crafted. Can you conclude from the crafted field value what was the tool used? In most cases yes, because the tools have their own unique fingerprint that give away their identity. It is not really matter which ICMP Query message type you generate with that tool. With manipulation of some fields, there is only one tool

