

Scan 21, June-2002

Bo Adler

BSI, www.fastcoder.net

thumper@alumni.caltech.edu

1. Listing of Files Submitted

Table 1. Core Files of Submission

File	Contents
index.html	This file, listing all files submitted
timestamp.html	Digital timestamp for this submission
analysis.html	Procedure used during investigation.
answers.html	Answers to HoneyNet questions.
Makefile	A makefile used during the process of creating the submission and timestamping it.
README	Describes the useful targets in <code>Makefile</code> .
timestamp.pl	A perl script to automate the process of getting a digital timestamp for our submission. Once the timestamp is received by email, this script is used to merge it into <code>timestamp.html</code> . It is also able to verify this information.

The following files are included in `files.tar`. They were generated during the analysis process.

Table 2. Files Generated During Analysis

File	Contents
udp-packets.txt (files/udp-packets.txt)	tcpdump output of just the udp packets found in <code>0215@000-snort.log</code> .
flurry-packets.txt (files/flurry-packets.txt)	tcpdump output of the specific udp packets which are the subject of this scan's questions.
note.gif (stylesheet-images/note.gif)	Images used in this document.

2. Timestamp Information for Submission

This file contains a digital timestamp certificate of a listing of the MD5 checksums of all files included in this submission (except `timestamp.html`). This timestamp is provided by Stamper (<http://www.itconsult.co.uk/stamper.htm>), a free timestamping service:

```
#REPLACE#
```

3. Analysis

3.1. Download and Verification

To begin analysis, I downloaded the gzip'd tar file and verified that the MD5 signature matched the one listed at the download page:

```
csh% wget http://project.honeynet.org/scans/scan21/0215@000-snort.log.tar.gz
--00:06:59-- http://project.honeynet.org/scans/scan21/0215%40000-snort.log.tar.gz
           => '0215@000-snort.log.tar.gz'
Connecting to project.honeynet.org:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 81,060 [application/x-tar]

   0K -> ..... [ 63%]
  50K -> ..... [100%]

00:07:00 (85.03 KB/s) - '0215@000-snort.log.tar.gz' saved [81060/81060]

csh% md5sum 0215@000-snort.log.tar.gz
58abd0cb0cbe4c31930225dd229352a5 0215@000-snort.log.tar.gz
csh% tar -tzf 0215@000-snort.log.tar.gz
0215@000-snort.log
csh% tar -xzf 0215@000-snort.log.tar.gz
```

At this point, a snort capture file is now available in the current directory.

3.2. Initial Analysis

The questions about this incident revolve around the flurry of UDP packets described in the introduction to the scan, so I decided that the first step should be to identify the contents of the packet capture and find the UDP packets in question.

```
csh% /usr/sbin/tcpdump -x -r 0215@000-snort.log | less
[...output deleted...]
```

I was disappointed to find that there was a great many packets within the capture file, and even limiting the listing to only UDP packets produced several different candidates for the "flurry" that the questions were about.

A clue in the introduction is that the packets were received on the evening of 15-Feb, so I decided to look at the timestamps of the packets to narrow the choices down:

```
csh% /usr/sbin/tcpdump -tt -r 0215@000-snort.log udp > files/udp-packets.txt
[...review the output for blocks of time...]
csh% /usr/bin/perl -e 'print scalar(gmtime(1013739967)), "\n";'
Fri Feb 15 02:26:07 2002
csh% /usr/bin/perl -e 'print scalar(gmtime(1013771265)), "\n";'
Fri Feb 15 11:07:45 2002
csh% /usr/bin/perl -e 'print scalar(gmtime(1013787598)), "\n";'
Fri Feb 15 15:39:58 2002
csh% /usr/bin/perl -e 'print scalar(gmtime(1013791658)), "\n";'
Fri Feb 15 16:47:38 2002
csh% /usr/bin/perl -e 'print scalar(gmtime(1013800942)), "\n";'
Fri Feb 15 19:22:22 2002
csh% /usr/bin/perl -e 'print scalar(gmtime(1013817015)), "\n";'
Fri Feb 15 23:50:15 2002
```

The last two groups of packets are clearly in the evening time, and looking at them more closely it became obvious to me that the incident questions were referring to the last group of packets:

```
csh% /usr/sbin/tcpdump -vv -x -r 0215@000-snort.log udp and src net 213.68.213 > files/flurry-packets
```

As suggested by the problem introduction, this flurry of packets appeared mystifying to me. There are a few facts that can be observed, however...

source IP from same subnet: The source IPs of the packets are all in the range 213.68.213.(130-144).

Potential inferences: most likely, all the source IPs are on a single LAN. This suggests that the attacker is expecting some response that they can detect without giving away their precise IP location.

destination IPs are sequential: Nine packets are received, sequentially addressed to IPs 172.16.1.(101-109) within the Honeynet.

Potential inferences: IP numbers 101 through 109 cannot normally form a valid subnet by themselves; the number of valid IPs in a subnet is a power of two (less two, for network and broadcast addresses). For this reason, I am surprised that the packets stop at 172.16.1.109, since there should be a few other valid IPs in the subnet. It seems likely that the attacker has done this on purpose, rather than a weird segmenting of subnet traffic for the Honeynet.

source and destination ports are apparently random: *Potential inferences:* The attacker is not trying to communicate with a traditional UDP service, or is perhaps trying to mask such communication. (If a slave process has been installed on a compromised machine in the Honeynet, it could be promiscuously sniffing the network for such communication, however.)

UDP checksum is not used: The UDP checksum on each packet is set to zero.

A Google search on the terms "*udp, checksum, 0*" produces a nice explanation (<http://www.ripe.net/ripe/mail-archives/dns-wg/19960501-19960601/msg00000.html>) of the value 0x0000 being used to indicate that the checksum is not used.

UDP payload is constant: The data segment of all nine UDP packets is the same 5 bytes: 44 4f 4d 02 00.

Potential inferences: These five bytes do not seem enough for any significant communication. If there is any communication intended by these packets, it might possibly be encoded in the port numbers or other fields of the IP header.

IP ttl fields are inconsistent: The ttl field of the IP header of nine packets varies from 151 to 226, with only one repeat.

Potential inferences: The close timing of packet reception makes it extremely unlikely that 8 of the nine packets traversed different routes to result in such dramatic differences in the ttl field. This implies that there is some value in varying that field, possibly to encode information for a slave process on a compromised machine.

3.3. The Bigger Picture

The scan questions (particularly Q5) imply that these UDP packets are merely probes, rather than communication with some compromised machine. I decided to load up ethereal and view the capture log (done by selecting File → Open and loading 0215@000-snort.log), to get some context for these mysterious packets.

By filtering for udp packets, selecting the last packet, and then resetting the filter, I was able to see that the flurry of UDP packets occurs very near the end of the packet capture. The timestamps indicate that the capture went on for 10 minutes past the flurry, which should be more than enough time for any responses to be generated.

The first thing that caught my eye about the last 10 minutes of packets is that there is IRC traffic between hosts 172.16.1.102 and 64.224.118.115. The traffic appears to be normal for this network, and there doesn't appear to be any additional data beyond the normal PING/PONG part of the IRC protocol.

Also notable was the fact that there are no ICMP errors appearing in `0215@000-snort.log`. This could be a simple attempt to map out the Honeynet network, which failed because of ICMP filtering at the Honeynet site. Running a filter on `"icmp"` shows a similar block of nine ICMP ECHO REQUEST packets going to hosts 172.16.1.(101-109), with no replies.

The similar block of nine ICMP pings piques my curiosity, because of the subnet issue mentioned previously. There are several other similar cases, including simultaneous connection attempts to port 53 (DNS) over all nine hosts. There are no packets outside of the 101-109 range, which leads me to discount the theory that the attacker had some reason for selecting only those hosts -- there appear to be multiple probes from multiple attackers within the packet capture, so it doesn't seem likely that they would all restrict themselves to the same range.

This leaves me believing that this is a simple probe trying to map out valid hosts in the Honeynet by eliciting ICMP responses. Pondering question 4 (are the packets from a single machine) leaves me unsatisfied, however. It bothers me that the `ttl` fields are varied, since a simple UDP probe shouldn't require that. I was unable to find anything on the web that might explain such behaviour.

3.4. Tools Used

During the analysis of this scan, we used the following tools:

Tool	Description	Location
md5sum	Calculates a checksum of a file. Often used to verify downloads.	standard
perl	A scripting language with powerful facilities for text manipulation.	http://www.cpan.org
tcpdump	Used to capture packets from the network, or to read previously saved packets.	http://www.tcpdump.org
ethereal	A packet analyzer, similar to tcpdump but equipped with a GUI and several additional features.	http://www.ethereal.com

4. Answers

4.1. Q1 Answer

It appears that the attacker is attempting to map out valid hosts on the Honeynet network, and potentially fingerprint which operating systems are in use.

4.2. Q2 Answer

The probing is achieved by sending UDP packets to invalid ports. This typically generates ICMP "port unreachable" packets sent back to the source IP of the UDP packet, enabling an attacker to detect the presence of a host. While many sites filter out ICMP "ping" packets to protect against mapping, they do not always filter TCP and UDP packets, allowing the attacker to determine valid hosts within the network.

4.3. Q3 Answer

Using random ports and source IP addresses makes it very difficult for a NIDS system to correlate the packets as being part of a network scan.

4.4. Q4 Answer

Given the timing of the nine packets, they must have been generated by a single machine. Almost every packet follows the previous packet by under 0.0008 seconds, while the IRC ping/pong packets are separated by over 0.0015 seconds (the pong reply in the IRC protocol is intended to occur "right away"). This data leads me to conclude that it's not credible to believe that multiple machines generated these packets.

4.5. Q5 Answer

Any ICMP errors generated by the packets would be sent back to the 213.68.213 network. The attack can observe these responses by promiscuously sniffing the network from any one machine in that LAN (assuming that a hub was in use).

4.6. Bonus Question Answer

Normally, yes. A tool like Xprobe (<http://www.sys-security.com/html/projects/X.html>) can fingerprint several operating systems on the basis of a single ICMP error reply to a UDP packet. On this Honeynet,

however, it appears that ICMP packets are filtered out, so that such a probe would produce no results.