snort and nmap – two sides of the same coin

# CAIN AND ABEL

RALF HILDEBRANDT

**nmap is a port scanner, which can search a target computer for open ports, and thus for potential security loopholes. Snort's task is to counteract nmap. Snort is a daemon which scans through a network for suspect packages and logs them.**

There are two sides to the world of network security – good and bad intentions. You're either wearing the white hat or black hat. Usually, there's a gaping chasm in between the two. Yet at the same time they are more similar than most people would like to admit. In any case they use the same tools.

Every security-conscious network administrator will want to make use of a port scanner one day and every hacker will be aware of the basic insecurity of every TCP/IP network. This is why many of them have a Network Intrusion Detection System monitoring the LAN. In this article we are going to take a closer look at two typical and frequently tried and tested examples from each genre.

## nmap – the dark side

*nmap* (for "Network Mapper") was developed by the hacker "Fyodor" primarily to scan large


networks. It does, of course, also work for individual hosts. Because of the large number of possible scans and options, there are now almost always several ways of going about the same thing. Sometimes you need a "fast" scan, sometimes you want to leave only minimal traces on the target system. There again, it might be necessary to get round a firewall, or scan for various protocols.

## nmap details

The "TCP three-way-handshake", which forms the foundation for all TCP connections, can be seen in Figure 1. A connection is initiated by a *SYN* packet. The other party responds with a *SYN/ACK* packet, to which the party initiating the connection must then respond with an *ACK* in order to complete the connection. The types of scan supported by nmap are:

- **TCP-connect() scan**
  Here the *connect()* system call of the operating system is used, thus a connection is made in a completely normal way. This is also the only type of scan which does not require any *root* privileges.
- **TCP-SYN ("half open") scan**
  In this case a *SYN* packet is sent to the corresponding port. If the response from the

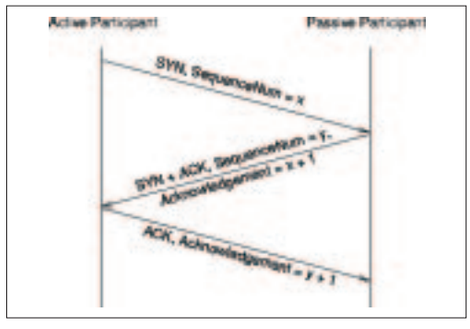


FIG 1: TCP three-way-handshake

target port is in the form of a packet with *SYN* and *ACK*, then the port is open. In this case an *RST* is sent, in order to prevent the connection being made in the first place (hence the name "half open"). An *RST* and *ACK* as response on the other hand characterises a closed port.

- **TCP-*FIN*, Xmas or NULL (stealth) scan**
  The fundamental idea behind a *FIN* scan is that a closed port responds to a *FIN* packet with an *RST*. Open ports, on the other hand, tend simply to ignore the packet. This behaviour is necessary (in accordance with RFC 793) for the correct functioning of the TCP.
  But even if a port is blocked by a packet filter, one may still be sent back an *RST* packet – in this case the *FIN* scan is wrongly reporting that any number of ports are open.
  A few systems (such as Microsoft) always send an *RST* regardless of the status of the ports. Therefore, with this type of scan, it is very easy to differentiate Unix and NT machines.
  In the case of a "full" *Xmas* scan all pre-defined flags (*FIN*, *SYN*, *RST*, *PSH*, *ACK* and *URG*) are set. The packet is decorated like a Christmas tree (hence the name). A "simple" *Xmas* scan has only *FIN*, *PSH* and *URG* set. According to RFC 793 the target system ought to send back *RST* for every closed port.
  With a *NULL* scan all the flags are cancelled (not set). According to RFC 793 the target system ought to send back *RST* for every closed port.
- **TCP ftp proxy (bounce attack) scan**
  This scan scarcely means anything any more because it is based on a feature of the ftp protocol which has by now been deactivated on most servers.
  In this case, a weakness of the FTP protocol was exploited. Details can be found at *http://www.insecure.org/nmap/hobbit.ftpbounce.txt*
  The user of this scan remains hard to locate for the scanned computer, as he/she is, as it were, hiding behind an FTP server which has a read-write access directory (for example */incoming*) and which offers the *proxy* feature. Phrack 51 lists as possible server types *wu-2.4(3)*, *wu-2.4(11)* and *FTP server SunOS 4.1*.
- ***SYN/FIN* scan with very small, fragmented packets**
  Instead of sending packets directly, they are split up into small IP fragments ("fragmented"). In this way, the TCP header is spread over several packets, so that it is harder for packet filters to detect exactly what is going on. Obviously this method cannot be used against firewalls, which collect and then defragment the IP fragments (as occurs for example through the kernel option *CONFIG_IP_ALWAYS_DEFRAG* under Linux).
- **TCP-*ACK*/Window scan**
  With the aid of this scan it is possible, for example, to determine whether a firewall is a simple packet filter for incoming SYN packets or a "stateful firewall". This scan sends an *ACK* packet with a random sequence number to the port. If an *RST* comes back, the port is classed as "unfiltered". If nothing (or an ICMP unreachable error) comes back, the port is classed as "filtered". Open ports cannot be detected.
- **TCP-Window scan**
  As with the *ACK* scan, only an anomaly in the *TCP window size reporting* code of an operating system is exploited. This means that in addition to the TCP-*ACK* scan, open ports can also be detected.
- **UDP raw ICMP port unreachable scan**
  This scan is fiddly, as open ports with UDP do not have to send any confirmation to our packet (UDP is not connection-oriented), nor closed ports an error packet. Fortunately most machines send an *ICMP_PORT_UNREACH*-error message, if a packet is sent to a closed UDP Port. So at least it is possible to find out if a port is closed.
  There are no guarantees that error messages for either UDP packets or ICMP will arrive. Therefore, a UDP scanner has to retransfer any potentially lost packets Otherwise, one would receive any amount of false positive results – say open ports where there simply are none.
  The scan is also unspeakably long-winded, as RFC 1812, Section 4.3.2.8 ("Rate Limiting") must be observed. This section stipulates the number of *ICMP error messages* per unit of time. So for example the Linux kernel in *net/ipv4/icmp.h* limits the creation of *ICMP destination unreachable* messages to 80 per 4 seconds, with a 0.25 second forced break, if this limit is exceeded.
- **ICMP echo scan (ping-sweep)**
  'Not really a port scan, as ICMP does not recognise any ports. In this case all machines are simply pinged in order to determine whether they are "up" or "down".
- **TCP-Ping scan**
  This scan sends a packet with *ACK* flag to a port (standard: 80). If, in response, an *RST* packet comes, the machine is "up". This scan can be used as an alternative when (as at *www.microsoft.com* for example) the echo port has been deactivated.
- **Direct (non-portmapper) RPC-scan**
  This scan functions in tandem with the scan types from *nmap*. All open ports are bombarded with SunRPC-NULL commands in order to detect whether they are RPC ports and if so, which program and version they are using.
- **Protocol scan**
  This scan sends IP headers (without data) with different protocol fields to the host. The host then (normally) returns a "Protocol Unreachable", for the protocols that it does not control. *nmap* can now create a list of the protocols supported by a process of elimination. This is very similar to the UDP scan from a design point of view. Naturally, there are also hosts here which do not send back

a "Protocol Unreachable" – in this case all protocols appear as "open". So this scan will not work against HP-UX Version 10.20 for instance.

- **Determination of an operating system using TCP/IP "Fingerprinting"**
  In this method, a selection of packets are sent with different TCP flags (*SYN*, *BOGUS*, Don't-Fragment-Bit...) to an open and a closed port respectively and the response is compared with entries from known systems from a databank (like a fingerprint). An overview of all flags specified in the protocol is given in Table 1. Obviously one can also modify the "fingerprint" of the TCP/IP stack on commercial Unixes. In particular, HP-UX 10.20 therefore comes with very naive default settings; this is where the tool *nettune* can work wonders:

```
/usr/contrib/bin/nettune -s tcp_random_seq 2
/usr/contrib/bin/nettune -s hp_syn_protect 1
/usr/contrib/bin/nettune -s ip_forwarding 0
echo 'ip_block_source_routed/W1' | \
/usr/bin/adb -w /stand/vmunix /dev/kmem
```

  This makes the creation of the TCP sequence numbers "more random", protection against SYN flooding is activated, IP forwarding is deactivated and source-routed packets are blocked by a direct kernel hack. These are TCP packets which are allocated their path through the network, the route, at the point of creation by the IP stack. But nowadays very few routers still accept this mechanism.
- **TCP Reverse-Ident scan**
  In 1996, Dave Goldsmith found in a posting on Bugtraq that the ident-protocol (RFC 1413) reveals the user under which a process connected via TCP is running. And this happens even if the process concerned did not even make the connection! That means that one can make a connection to the HTTP port and find out using *identd* under which user the *httpd* is running. This scan functions only with a TCP-connect() scan (Option *-sT*).
  In addition, *nmap* offers performance and reliability features such as a dynamic calculation of delay times, packet time-outs and re-transmission attempts, parallel port scans and recognition of machines which are switched off by means of parallel pings.
  Naturally, one can easily specify the hosts and ports to be scanned. Apart from stating the port status, *nmap* also defines the predictability of the TCP sequence numbers and thereby the susceptibility of the machine to IP spoofing attacks.

**Table 1: TCP-Flags and what they mean**

| TCP-Flag | Meaning |
|---|---|
| FIN | finish disconnect |
| SYN | synchronize |
| RST | reset |
| PSH | push |
| ACK | acknowledge |
| URG | urgent |
| (2) | reserved |
| (1) | reserved |

## Hiding your own IP address

Furthermore, it is also possible to activate a so-called "decoy" option (*-D*). This prevents the other party from finding out which host has initiated the scan. By specifying the option *Ddecoy1.host.com,ME,decoy2.host.com*, *nmap* "forges" packets with the sender addresses of *decoy1.host.com* and *decoy2.host.com*.

If both *decoy1.host.com* and *decoy2.host.com* are "up", these will send *RST*-packets as expected, so that for the target machine *decoy1.host.com*, *decoy2.host.com* and the local host can be distinguished. If the *decoy*-Hosts are "down", the target of our scan will be flooded with *SYN*-packets.

In standard mode *nmap* uses an ICMP ping and a TCP-*ACK* ping with Port 80 as originating port (Port 80 is often let through firewalls because of HTTP-requests) in order to determine whether machines are "up". Then the port scan is performed. An attempt is made to define the operating system of the scanned host as a last resort.

It is obvious that one could easily write a rule for an Intrusion Detection System (IDS), which would detect an *nmap*-scan with certainty. Therefore one could, with *-P0* for example, deactivate the ICMP ping. Explicit activation of the TCP ping is done through *-PT*.

## Examples

```
% nohup nmap -r -iR -I -sT -p53 > named.scan.out &
% tail -f named.scan.out
```

Now we can go on the hunt for machines on which *named* is running as *root*. The option *-r* scans the ports of the target machine in a random sequence, *-iR* selects random IPs as target for the scan, *-I* activates the reverse ident scan, which only functions with a TCP-connect()scan (*-sT*). *-p53* finally defines Port 53 as scan target.

In the second example, we want to scan *target.host*, but at the same time avoid being detected too easily. So we will use a few decoy hosts:

```
% nohup nmap -r -P0 -sS -Ddecoy1,decoy2,decoy3,decoy4,decoy5 target.host
```

In this instance, the hosts *decoy1* to *decoy5* should exist and be reachable or "up". The option *-P0* deactivates the *pings* from *target.host* before the scan – we are assuming that it is "up". *-sS* activates the SYN-scan and *-Ddecoy1,decoy2,decoy3, decoy4,decoy5* uses the hosts decoy1 to decoy5 in order to create a bit of confusion.

## Snort – looking on the bright side


Snort (*http://www.snort.org/*) from Martin Roesch is a so-called Intrusion Detection System (IDS). It is capable of analysing IP-Network traffic online and recording packets.

It can also be used for protocol analysis, as well as for looking into the flow of network data for contents and logging corresponding packets together with their contents.

By using context-sensitive rules, Snort can be used to detect a multitude of attacks and scans, such as for example Buffer Overflows, Stealth Port Scans, CGI Attacks, SMB probes and active OS Fingerprinting and to report these to the administrator.

Snort can – if it has been configured with *--enable-flexresp* – even respond to incoming packets, for example by sending *RST*-packets, which are intended to close the connection down.

This reporting can be done via *syslog()*, a file, a UNIX Domain Socket or *smbclient* (in the form of a WinPopup Requester).

## Philosophy

Intrusion Detection Systems (IDS) are technologies which reduce the risk of intrusion, but do not eradicate it altogether.

An attack is a transient incident. Conversely, a "vulnerability" (weak point) permanently contains within itself the risk of an attack. The difference between an attack and a vulnerability, is that the attack exists only at a specific time, while the vulnerability exists regardless of the time of observation. One could also say that an attack represents an attempt to exploit a weak point.

## What do I need?

Snort is based on libpcap, the Packet Capturing Library. This can be obtained from *http://www.tcpdump.org/*. When using *--enable-flexresp* it is necessary to install the *libnet* library from *http://www.netfactory/libnet*.

Last of all, of course, you need rules, too. These can also be found at *http://www.snort.org* – click there on "Rule Database". It is advisable to copy these to */etc/snort.conf*, as they more or less represent the "configuration" of Snort.

## Things to look out for?

These rules are obviously based on so-called signatures, which must be known in advance. This means that our IDS (like all others) does include a risk of false alarms. We must differentiate between two sorts of false alarms:

- *false positives*: normal network activity is classed as an "attack";
- *false negatives*: A real attack is ignored.

Consequently, we still need a human being who will subject alarms to thorough investigations. *false negatives* are more dangerous than *false positives*, as they give the user a treacherous feeling of security. Equally, *snort* can even be used for a DoS (Denial of Service), for example by flooding log files (and thus the disks).

## What should I change?

In the rules which we have installed by this point under */etc/snort.conf*, a few alterations must be made.

```
preprocessor portscan: 10.0.0.1/8  \
   7 1 /var/log/snort/portscan.log
```

specifies how many connections (in this case, 7) per unit of time (in this case 1 second), to which target addresses (the entire *10.x.x.x* network) are classified as a port scan. The port scan is logged in */var/log/snort/portscan.log*.

In the line *var HOME_NET 10.0.0.1/24* the local, trustworthy network must be entered, especially as many rules differentiate between machines within and outside the *HOME_NET*. Here, this is *10.0.0.x*. An individual host *10.0.0.1* would consequently be *10.0.0.1/32*.

In the line

```
preprocessor portscan-ignorehosts: 10.0.0.1/2
30
```

it is possible to enter the hosts from which port scans should be ignored. There, for example, one could use the same settings as for *HOME_NET*.

## The start

It is advisable to activate *Snort* by means of a start script (for example */sbin/init.d/snort* when changing to a run level with network support. I start *Snort* from */sbin/init.d/snort* with:

```
snort -u snort -g snort -D -d -b -s -c /etc/s2
nort.conf -l /var/log/snort
```

## Brighter, better, faster!

The log files are at first glance truly enormous and there are numerous log file analysers which exist for *snort*, which statistically process the threats for the administrator.

## Rules

The format of the rule file is described at *http://www.snort.org/writing_snort_rules.htm*. A sample rule is:

*The parameters of snort:*

***-t /snort-chroot***
*snort can be allowed to run similar to BIND-8.x chrooted, in order to minimise the risk of snort being compromised.*
*To do this is it advisable to statically link snort (LDFLAGS=@LDFLAGS@ -s -static in the Makefile). Of course, the whole thing only makes sense when snort does not have root privileges:*

***-u snort***
*Start snort with the user-id "snort". As snort processes data from the network, we don't want it to have root privileges – or else an exploit by snort could lead to a root compromise. For that reason, a user "snort" and a group "snort" (to which pnly the user "snort" belongs) must be created. This user should be unable to log in and have no shell.*

***-g snort***
*Start snort with the group-id "snort".*

***-D***
*Start snort in daemon mode.*

***-d***
*Also, log the data of the application layer.*

***-b***
*Writes logged packets in tcpdump format on the disk. This is necessary for performance reasons, especially with 100 MBit networks!*

***-s***
*Write alarms into the syslog.*

***-c /etc/snort.conf***
*Path to configuration- and rule file*

***-l /var/log/snort***
*Path to the directory in which snort stores its log files.*

```
alert icmp $HOME_NET any -> !$HOME_NET any (m
sg:"IDS191 - DDoS - \
barbed wire server-response"; content: "|6
6 69 63 6B 65 6E|"; \
itype: 0; icmp_id: 667;)
```

## Rule Header

Every rule starts with a "rule header"; it consists of several fields:

```
alert icmp $HOME_NET any -> !$HOME_NET any
```

## Action ("rule action")

*alert*: a packet generates an alarm and is logged.
*log*: The packet is only logged.
*pass*: The packet is ignored.
In our example an alarm is produced.

## The protocol

In our example this is an ICMP-packet.

## IP addresses

IP addresses are stated in the form *w.x.y.z/n*, where *w.x.y.z* is an IP address and *n* is a CIDR block.
So *10.0.0.0/8* specifies the whole 10th Class-A subnetwork, and *10.0.0.0/24* merely all hosts from *10.0.0.1* to *10.0.0.254*.
As operator, the negation operator *!* is available.
The packets in our example come from *$HOME_NET*.

## Ports

Ports can be specified as individual ports, domains or negations. In that case, the key word *any* stands for any port you like:

```
1
Port 1

1024:
All ports above Port 1024 (inclusive)

:1024
All ports below Port 1024 (inclusive)

Our example involves ICMP packets, but ICMP d
oesn't know any ports, so we are using any.
```

## Directional operator

The directional operator states in which direction the rules apply:

**Table 2: Snort rule headers**

| Option | Meaning |
|---|---|
| msg | Generates an alarm and writes in the log. |
| logto | Log to a special file instead of the default file. |
| ttl | TTL field in the IP header |
| id | Fragment-ID field in the IP header |
| dsize | Size of the packet content |
| content | Content of a packet |
| offset | This can be used to specify an Offset for the content option, with which the content will be matched. |
| depth | This can be used to specify the maximum search length for the content option. |
| nocase | This is used to make the content option non-case sensitive. |
| flags | TCP-Flags |
| seq | TCP sequence number |
| ack | TCP acknowledgement field |
| itype | ICMP type |
| icode | ICMP code |

| Option | Meaning |
|---|---|
| session | This can be used to log the application layer information for a session. |
| icmp_id | ICMP-Echo ID field |
| icmp_seq | ICMP echo sequence number |
| ipoption | IP option fields: |
| rr | Record route |
| eol | End of list |
| nop | No op |
| ts | Time Stamp |
| sec | IP security option |
| lsrr | Loose source routing |
| ssrr | Strict source routing |
| satid | Stream identifier |
| rpc | Check RPC services for specific application or procedure calls |
| resp | This can be used to generate an active reply (for example to prevent disconnection by sending an RST packet). |

**->** On the left hand side of -> is the source, and on the right the target.

The bidirectional operator. Network traffic in both directions is acquired.

In our example the rule is applied to all ICMP packets leaving our network.

## Rule Options

After the rule header come the options for the rule, and an overview can be found in Table 2.

```
(msg:"IDS191 - DDoS - Barbed wire \
server-response"; content: \
"|66 69 63 6B 65 6E|"; itype: 0; icmp_id: 667;)
```

In our example, a warning is created (*msg:"IDS191 - DDoS - Barbed wire server-response";*), when the content of the packet contains the above-named sequence of bytes anywhere (*content: "|66 69 63 6B 65 6E|";*) and the ICMP type 0 is (*itype: 0;*) and the ICMP echo ID field has the value 667 *icmp_id: 667;*.

## Preprocessors

In snort there are a few preprocessors, which apply before the rules; there are:

- *preprocessor minfrag: 128*
  This preprocessor recognises fragmented packets under the specified fragment size (in this case: 128). Normally, packets are fragmented on their way from the source to the target by routers, but one may assume from this that no hardware fragments smaller than 512 Bytes are produced. So everything smaller is created artificially.
- *preprocessor http_decode: 80 8080*
  With this preprocessor, HTTP URLs can be converted into clear text ASCII.
- *preprocessor portscan: 192.168.1.0/24 Ports Time /var/log/portscan.log*
  More than "ports"; connections during the period of "time" seconds on the network *192.168.1.0/24*

A typical attack sequence, which *snort* has captured in the network of a customer, can be found in the box "Attacks and Strategies"

Explanations:

At 19:52:24, ???.???.86.226 started its "port scan" – as the maximum number of connections per unit of time had been exceeded, *Snort* has characterised the connections as port scans. The attacker could have dodged our port scan preprocessor by simply having a bit more patience, so *nmap* offers a "timing" option for the speed of a port scan: *nmap -T Paranoid* or *nmap -T Sneaky* might possibly not have triggered the preprocessor.

The speed of the attack, however, tends to indicate an automatic tool rather than *nmap* – in the end it was all over in about 3 seconds.

The scan succeeded with SYN-*FIN* packets on ports 53 of xxx.yyy.106.18 and xxx.yyy.106.23; the rule which captured this is:

```
alert tcp !$HOME_NET any -> $HOME_NET any (msg:"SCAN-SYN FIN";flags:SF;)
```

After that, for both machines, a test was performed to find out if the nameservers support *inverse queries* :

```
alert udp !$HOME_NET any -> $HOME_NET 53 (msg:"IDS277 - NAMED Iquery \
Probe"; content: "|0980 0000 0001 0000 0000|"; offset: "2"; depth: "16";)
```

This indicates that an attempt may be made to exploit a buffer overflow in BIND (*http://www.cert.org/advisories/CA-98.05.bind_problems.html*), which grants the attacker *root* privileges. For this to work, the server concerned must have the *fake-iquery* option activated.

Finally, the BIND version was queried:

```
alert udp !$HOME_NET any -> $HOME_NET 53 (msg:"IDS278 - NAMED Version \
Probe"; content: "|07|version|04|bind|00 0010 0008|"; nocase; offset: "13";depth: "32";)
```

The above buffer overflow can only be taken advantage of on BIND nameservers with version numbers lower than BIND 4.9.7 (BIND Version 4) and BIND 8.1.2 (BIND Version 8).

At 19:54:04 *snort* then reports the status of the port scan; there were 7 connections or attempted connections, spread over 3 machines on the sub-network being monitored, 5 of them TCP and 2 UDP.

**Attacks and Strategies**

```
Jun 25 19:52:24 snort[11597]: spp_portscan: PORTSCAN DETECTED from ???.???.86.226
Jun 25 19:52:24 snort[11597]: SCAN-SYN FIN: ???.???.86.226:53 -> xxx.yyy.106.18:53
Jun 25 19:52:24 snort[11597]: SCAN-SYN FIN: ???.???.86.226:53 -> xxx.yyy.106.23:53
Jun 25 19:52:25 snort[11597]: SCAN-SYN FIN: ???.???.86.226:53 -> xxx.yyy.106.25:53
Jun 25 19:52:26 snort[11597]: IDS277 - NAMED Iquery Probe: ???.???.86.226:1565 -> xxx.yyy.106.18:53
Jun 25 19:52:26 snort[11597]: IDS277 - NAMED Iquery Probe: ???.???.86.226:1568 -> xxx.yyy.106.25:53
Jun 25 19:52:27 snort[11597]: MISC-DNS-version-query: ???.???.86.226:1565 -> xxx.yyy.106.18:53
Jun 25 19:52:27 snort[11597]: MISC-DNS-version-query: ???.???.86.226:1568 -> xxx.yyy.106.25:53
Jun 25 19:54:04 snort[11597]: spp_portscan: portscan status from ???.???.86.226: 7 connections across 3 hosts: TCP(5), UDP(2) STEALTH
Jun 25 19:57:02 snort[11597]: spp_portscan: End of
portscan from ???.???.86.226
```

## Moral

BIND should – if at all – always be installed in the most up to date version. It is precisely the "professional" Unixes which are having problems with this because the patches come out relatively late. The nameserver should only respond to queries from the local network (but also, of course, to all queries which concern primary and secondary zones) hence */etc/named.conf* contains the following:

```
acl "trusted" {
   134.169.0.0/16;
   localhost;
};

acl "bogon" {
   0.0.0.0/8;     // Null address
   1.0.0.0/8;     // IANA reserved, popular fa➥
kes
   2.0.0.0/8;
   192.0.2.0/24;  // Test address
   224.0.0.0/3;   // Multicast addresses
          // The following Enterprise
networks may or may not be bogus.
   10.0.0.0/8;
   172.16.0.0/12;
   192.168.0.0/16;
};

options {
   allow-query {
      trusted;
      // only queries from "trusted" hosts
   };
   allow-recursion {
      trusted;
      // only recursive queries
from "trusted" hosts
   };
   allow-transfer {
      none;
      // no-one can have my zones!
   };
   blackhole {
      bogon;
      // I'm not talking to them
   };
};
```

In addition, the issue of "version.bind" should be prohibited (*dig @server version.bind CHAOS TXT*), which really does make sense in view of the increasing number of attacks on nameservers in recent times:

```
zone "bind" chaos {
   type master;
   file "master/bind";
};
```

And the zonefile *master/bind*:

```
$ORIGIN bind.
$TTL 1W

@      1D CHAOS SOA    localhost. root.local➥
host. (
                     1             ; serial
                     3H            ; refresh
                     1H            ; retry
                     1W            ; expiry
                     1D )          ; minimum
       CHAOS NS       localhost.
```

It would also be possible to fill the zone with "false" (version) data, in order to attract hackers. Alternatively, one could also install other nameservers such as for example *tinydns* from Dan Bernstein, which has been proven to be considerably more secure.

## The Attacker

Is the attacker also a victim? We use *nmap* to obtain information:

```
# nmap -T Sneaky -O -sS ???.???.86.226

Starting nmap V. 2.54BETA1 by fyodor@insec➥
ure.org (www.insecure.org/nmap/)
Interesting ports on ??????????.???????????➥
??.co.jp (???.???.86.226):
(The 1511 ports scanned but not shown below a➥
re in state: closed)
Port       State      Service
21/tcp     open       ftp
22/tcp     open       ssh
23/tcp     open       telnet
25/tcp     open       smtp
37/tcp     open       time
53/tcp     open       domain
70/tcp     open       gopher
98/tcp     open       linuxconf
109/tcp    open       pop-2
110/tcp    open       pop-3
111/tcp    open       sunrpc
113/tcp    open       auth
143/tcp    open       imap2
5680/tcp   open       canna

TCP Sequence Prediction: Class=truly random
             Difficulty=9999999 (Good luck!)

Remote operating system guess: Cobalt Linux
4.0 (Fargo)
Kernel 2.0.34C52_SK on MIPS or TEAMInternet S➥
eries 100 WebSense
```

## MTA

```
% telnet ???.???.86.226 25
Trying ???.???.86.226...
Connected to ???.???.86.226.
Escape character is '^]'.
220 ??????????.?????????????.co.jp ESMTP Sen➥
dmail 8.8.7/3.7W1.0; Tue, 27 Jun 2000 17:30:4➥
7 +0900
QUIT
221 ??????????.?????????????.co.jp closing c➥
onnection
Connection closed by foreign host.
```

See Bugtraq Vulnerability ID 717 and many others. But this query is somewhat lacking in subtlety. It's much more elegant to send mail to "foobar@??????????.?????????????.co.jp" and put it, for example, in postfix *debug_peer_list* = *??????????.?????????????.co.jp*; then the log process will be seen in */var/log/mail*.

## FTP Daemon

```
% ftp ???.???.86.226
Connected to ???.???.86.226.
220 ??????????.?????????????.co.jp FTP serve➥
r (Version wu-2.6.0(1) Thu Oct 21 12:22:27 ED➥
T 1999) ready.
```

A WU-FTPD. See Bugtraq Vulnerability ID 1387.

## Nameserver

```
% dig @???.???.86.226 version.bind CHAOS TXT
;; ANSWER SECTION:
VERSION.BIND.          0S CHAOS TXT    "8.1.2"
```

An ISC BIND 8.1.2. See Bugtraq Vulnerability ID 983.

## POP2/POP3 Daemon

```
% telnet ???.???.86.226 109
Trying ???.???.86.226...
Connected to ???.???.86.226.
Escape character is '^]'.
+ POP2 ??????????.?????????????.co.jp v4.5➥
1 server ready
QUIT
+ Sayonara
Connection closed by foreign host.
% telnet ???.???.86.226 110
Trying ???.???.86.226...
Connected to ???.???.86.226.
Escape character is '^]'.
+OK POP3 ??????????.?????????????.co.jp v7.5➥
9 server ready
QUIT
+OK Sayonara
Connection closed by foreign host.
```

POP2 is, exceptionally, invulnerable. See Bugtraq ID 283.

## IMAP Daemon

```
% telnet ???.???.86.226 143
Trying ???.???.86.226...
Connected to ???.???.86.226.
Escape character is '^]'.
* OK ??????????.?????????????.co.jp IMAP4rev➥
1 v12.250 server ready
```

A WU-IMAPD 4.7. See *http://oliver.efri.hr/~crv/security/bugs/Linux/imapd9.html*

The computer is open like a garden gate and badly patched. Most likely a victim itself – who knows what might be running on that one? An RPC and Ident-scan provides, in addition to our above findings, also:

```
% nmap -P0 -v -v -sT -sU -I -sR ???.???.86.226

... snip ...

(The 3067 ports scanned but not shown below a➥
re in state: closed)
Port       State      Service (RPC)         O➥
wner
21/tcp     open       ftp
22/tcp     open       ssh
23/tcp     open       telnet
25/tcp     open       smtp
37/tcp     open       time
37/udp     open       time
53/tcp     open       domain
53/udp     open       domain
70/tcp     open       gopher
98/tcp     open       linuxconf
109/tcp    open       pop-2
110/tcp    open       pop-3
111/tcp    open       sunrpc (rpcbind V2)
111/udp    open       sunrpc (rpcbind V2)
113/tcp    open       auth
143/tcp    open       imap2
514/udp    open       syslog
3130/udp   open       squid-ipc
5680/tcp   open       canna
```

## Canna

Also, *canna*, a service provider which converts Japanese Kana into Kanji characters, displays numerous vulnerabilities:
*http://www.securityfocus.com/bid/757*Bugtraq ID 757
*http://www.securityfocus.com/bid/758*Bugtraq ID 758
*http://packetstorm.securify.com/advisories/debian/debian.canna.txt*Debian Security Advisory
*http://packetstorm.securify.com/advisories/freebsd/FreeBSD-SA-00:31.canna*
FreeBSD-SA-00:31 ■


## The author

*Since completing his degree in computer science at TU-Braunschweig, Ralf Hildebrandt Ralf.Hildebrandt@innominate.de has been working as a systems engineer.*


### *Info*


*[1]http://www.snort.org/ Snort IDS*
*[2]http://www.insecure.org/nmap/ nmap network scanner*
*[3]http://www.securityfocus.com/ security information*
*[4]http://neworder.box.sk/ security information and exploits*
*[5]http://www.faqs.org/rfcs/rfc793.html RFC 793 TCP specification*
*[6]http://www.faqs.org/rfcs/rfc1413.html RFC 1413 identification protocol*
*[7]http://www.faqs.org/rfcs/rfc1812.html RFC 1812 Requirements for IP Version 4 Routers*
*[8]http://phrack.infonexus.com/ Phrack Phrack 51 describes scan techniques*
*[9]http://xanadu.rem.cmu.edu/snort/ Tools from Yen-Ming Chen:*
*[10]http://www.silicondefense.com/snortsnarf/ Silicon Defence*


■