

Hacker Tools and Their Signatures, Part One: bind8x.c

Toby Miller 2001-04-11

Hacker Tools and Their Signatures, Part One: bind8x.c

by *Toby Miller*

last updated April 11, 2001

Purpose

This article is the first in a series of papers detailing hacker exploits/tools and their signatures. This installment will examine the Berkley Internet Name Domain exploit bind8x.c. The discussion will cover the details of bind8x.c and provide signatures that will assist an IDS analyst in detecting it. This paper assumes that the reader has some basic knowledge of TCP/IP and understands the tcpdump format.

BIND and DNS

"DNS is a distributed database that is used by TCP/IP to map between hostnames and IP addresses." [1] In short, this means when a user types in www.securityfocus.com on his/her browser, the computer does not automatically know what the user is requesting. The user's computer then goes out to the DNS server that was assigned to it and asks the DNS server to tell it the IP address of the requested URL. This also holds true for resolving IP addresses to hostnames. A good example of what really happens is shown in Figure 1. Figure 1 shows a TCPDUMP read-out from my home computer as it talks to its DNS server.

```

22:12:51.704487 > my.computer.net.1025 > my.isp?s.DNS.domain: 23838+A?
www.securityfocus.com. (39) (ttl 64, id 217)
    4500 0043 00d9 0000 4011 8f3a xxxx xxxx
    xxxx xxxx 0401 0035 002f 12ea 5d1e 0100
    0001 0000 0000 0000 0377 7777 0d73 6563
    7572 6974 7966 6f63 7573 0363 6f6d 0000
    0100 01
22:12:51.884600 > my.isp?s.DNS.net.domain > my.computer.net.1025:23838 q:
www.securityfocus.com. 1/1/1 www.securityfocus.com. A www.securityfocus.com
(89) (ttl 55, id 2916)
    4500 0075 0b64 0000 3711 8d7d xxxx xxxx
    xxxx xxxx 0035 0401 0061 62e9 5d1e 8180
    0001 0001 0001 0001 0377 7777 0d73 6563
    7572 6974 7966 6f63 7573 0363 6f6d 0000
    0100 01c0 0c00 0100 0100 0129 9100 0442
    2697 0ac0 1000 0200 0100 020b cd00 0603
    6e73 32c0 10c0 4300 0100 0100 00ba 4c00
    0442 2697 02
22:12:51.887833 > my.computer.net.1050 > www.securityfocus.com.www S
1691535048:1691535048(0) win 32120 <mss 1460,sackOK,timestamp 464785

```

```
0,nop,wscale 0> (DF) (ttl 64, id 218)
      4500 003c 00da 4000 4006 9f04 xxxx xxxx
      4226 970a 041a 0050 64d2 c6c8 0000 0000
      a002 7d78 e813 0000 0204 05b4 0402 080a
      0007 1791 0000 0000 0103 0300
```

Figure 1: DNS Communications

By looking at the first packet we can see that the destination port is 53 and that the packet itself is a UDP packet. I think that part of the packet is pretty clear. It's the DNS part of the packet that we need to make sense of. Before we look at DNS lets look at the header for DNS in Figure 2.

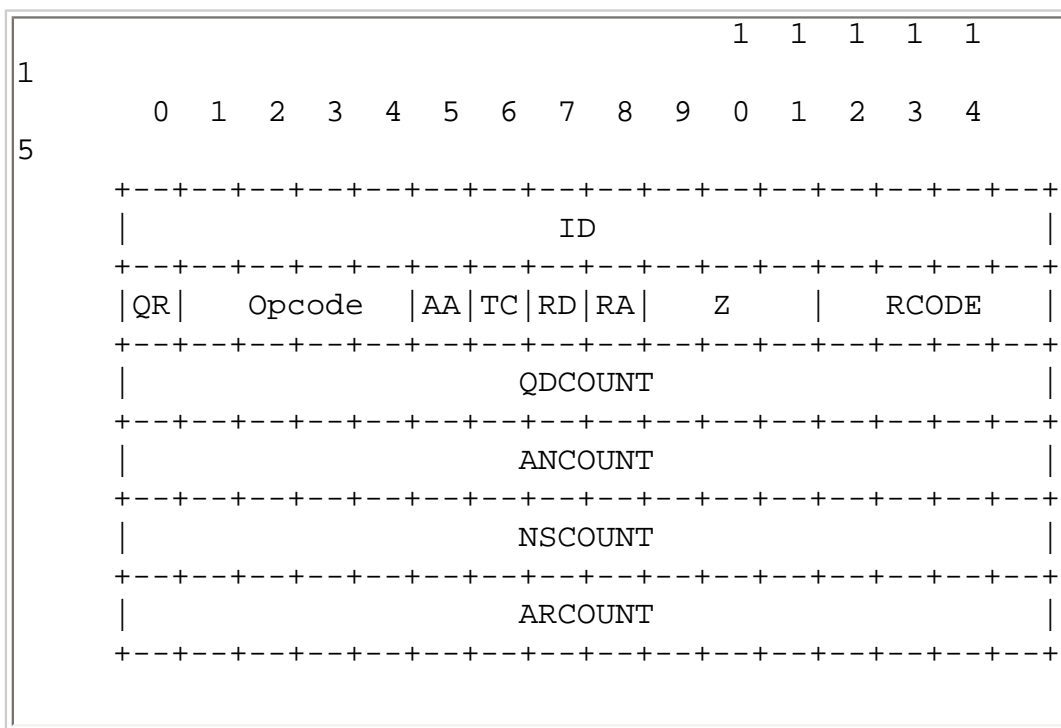


Figure 2: DNS Header Format [2]

Figure 2 gives us a good idea at what we are looking at in the packet. From this header we can see that 23838(hex 5d1e) is the identification number in the DNS packet. The ID number is sent to the server by the client and is then returned to the client by the server. The purpose of this is to allow the client to match responses with requests.

The next item we want to look at is the + sign. In the math world this is the addition sign; however, in the DNS world it means that the RD (recursion desired) flag is set. The 'A' means that we want an IP address.

Packet number 2 only has one area that needs explaining. That is the letter q - possibly meaning a query.

Finally, in the third packet we see that my computer was able to obtain the IP address of www.securityfocus.


```

0000 0000 0000 0000 0000 0000 0000 0000
0000
16:01:44.773774 < 192.168.1.5.1042 > 192.168.1.25.domain: 57005+ [b2&3=0x180]
[7q] [1au] (510) (ttl 64, id 4627)
4500 021a 1213 0000 4011 e351 c0a8 0105
c0a8 0119 0412 0035 0206 b5c3 dead 0180
0007 0000 0000 0001 3f00 0102 0304 0506
0708 090a 0b0c 0d0e 0f10 1112 1314 1516
1718 191a 1b1c 1d1e 1f20 2122 2324 2526
2728 292a 2b2c 2d2e 2f30 3132 3334 3536
3738 393a 3b3c eb0a 0200 00c0 0000 0000
003f 0001 eb44 5e29 c089 4610 4089 c389

```

Figure 3: TCPDUMP of Bind8x.c

What's so interesting about these packets? First, we see that the packets are inverse queries. What are inverse queries? Inverse queries are the reverse mappings of standard query operations[3]. Another interesting item we want to look at is '48879', which is the ID. If you look closely at the packet we see that 48879 = beef (great signature!). By looking at 0980 in the first packet we also see a few items of interest.

Figure 4 is a print out from ethereal:

```

Domain Name System (query)
Transaction ID: 0xbeef
Flags: 0x0980 (Inverse query)
0... .. = Query
.000 1... .. = Inverse query
.... ..0. .... = Message is not truncated
.... ...1 .... = Do query recursively
.... .... ..0 .... = Non-authenticated data is
unacceptable
Questions: 0
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
Answers
<Name goes past end of captured data in packet>:
type unused, class unknown
Name: <Name goes past end of captured data
in
packet>
Type: unused
Class: unknown
Time to live: 0 time
Data length: 0
Data

```

Figure 4: Inverse Query Packet Read Out

Figure 4 shows the normal DNS items. But let us look at 'Answers'. 'Answers' shows us that the Name goes past end of captured data in packet. This is where the overflows are beginning. The next packet is the victim's reply. To sum it up in a few short words it's saying "Name goes past end of captured data in packet>: type unused, class unknown". In other words, it has no clue what's going on.

Lets take a look at the ethereal output for the next packet.

```

Domain Name System (query)
  Transaction ID: 0xdead
  Flags: 0x0180 (Standard query)
    0... .. = Query
    .000 0... .. = Standard query
    .... ..0. .... = Message is not truncated
    .... ...1 .... = Do query recursively
    .... .... ...0 .... = Non-authenticated data is
unacceptable
  Questions: 7
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 1
  Queries
    <Name contains a pointer that goes past the end of
the packet>: type unused, class unknown
      Name: <Name contains a pointer that goes past
the end of the packet>
      Type: unused
      Class: unknown
    <Name goes past end of captured data in packet>:
type A, class unknown
      Name: <Name goes past end of captured data in packet>
      Type: Host address
      Class: unknown

```

Figure 5: Ethereal Output

Once again we can look inside the packet and come up with a great signature. If you look at the ID you will see 'dead'. Remember 57005 == dead. This time though, look at how many questions were sent over - seven (7). These little items are great if you are trying to develop IDS signatures for specific "hacker" tools. The reply to that packet looks like this.

```

16:01:44.774845 > 192.168.1.25.domain > 192.168.1.5.1042: 57005
[7q] q:
^@^A^B^C^D^E^F^G^H^I^J^K^L^M^N^O^P^Q^R^S^T^U^V^W^X^Y^Z^[^\]^_
!"#$%&'()*+,-./0123456789:;<M-k^J.^@^@. 0/0/1 (533) (ttl 64,
id 3)

      4500 0231 0003 0000 4011 f54a c0a8 0119
      c0a8 0105 0035 0412 021d 6fbc dead
8180

      0007 0000 0000 0001 3f00 0102 0304 0506
      0708 090a 0b0c 0d0e 0f10 1112 1314 1516
      1718 191a 1b1c 1d1e 1f20 2122 2324
2526

      2728 292a 2b2c 2d2e 2f30 3132 3334 3536
      3738 393a 3b3c eb0a 0200 00c0 0000 0000
      003f 0001 eb44 5e29 c089 4610 4089 c389
      460c

```

Figure 6: TCPDUMP Output of .25's Reply

The next couple of packets we see here show us 192.168.1.5 making a connection with 192.168.1.25. Then we will see 192.168.1.25 give .5 root access (the gold). The initial connection is shown in Figure 7.

```

16:01:44.789072 <: 192.168.1.5.1373 > 192.168.1.25.36864: S
3252931087:3252931087(0) win 32120 <mss 1460,sackOK,timestamp
1741083 0,nop,wscale 0> (DF) (ttl 64, id 4629)

16:01:44.790474 > 192.168.1.25.36864 > 192.168.1.5.1373: S
3700394557:3700394557(0) ack 3252931088 win 32120 <mss
1460,sackOK,timestamp 33477 1741083,nop,wscale 0> (DF) (ttl 64,
id 4)

16:01:44.790849 < 192.168.1.5.1373 > 192.168.1.25.36864: . 1:1(0)
ack 1 win 32120 <nop,nop,timestamp 1741083 33477> (DF) (ttl 64,
id 4630)

```

Figure 7

Notice the high port (36864) that 192.168.1.5 is trying to connect to. To save some space I will omit ACK packets, unless they are vital to the sequence of events. Here comes root!!

```

16:01:44.809078 < 192.168.1.5.1373 > 192.168.1.25.36864:
P
1:16(15) ack 1 win 32120 (DF)
(ttl 64, id 4631)
      4500 0043 1217 4000 4006 a52f c0a8 0105
      c0a8 0119 055d 9000 cle3 ca10 dc8f 8a3e
      8018 7d78 90b5 0000 0101 080a 001a 911d
      0000 82c5 756e 616d 6520 2d61 3b20 6964
      3b0a 00

```

Figure 8

The bolded hex numbers in bold come out to 'uname -a; id;'. Again, notice the high destination port.

```

16:01:44.912351 > 192.168.1.25.36864 > 192.168.1.5.1373: P
1:73(72) ack 16 win 32120 <:nop,nop,timestamp 33490 1741085>
(DF)
(ttl 64, id 6)
4500 007c 0006 4000 4006 b707 c0a8 0119
c0a8 0105 9000 055d dc8f 8a3e cle3 calf
8018 7d78 9af8 0000 0101 080a 0000 82d2
001a 911d 4c69 6e75 7820 7374 6565 6c65
7273 3132 2032 2e32 2e31 322d 3230 2023
3120 4d6f 6e20 5365 7020 3237 2031 303a
3235 3a35 3420 4544 5420 3139 3939 2069
3538 3620 756e 6b6e 6f77 6e0a

```

Figure 9

Note that .25 is now sending all of its computer information to .5.

```

16:01:44.944313 > 192.168.1.25.36864 > 192.168.1.5.1373: P
73:97(24) ack 16 win 32120 (DF)
(ttl 64, id 7)
      4500 004c 0007 4000 4006 b736 c0a8 0119
      c0a8 0105 9000 055d dc8f 8a86 cle3 calf
      8018 7d78 bd97 0000 0101 080a 0000 82d5
      001a 9127 7569 643d 3028 726f 6f74 2920
      6769 643d 3028 726f 6f74 290a

```

Figure 10

Ok, we now have root. The bolded hex numbers above = uid=0(root) gid=0(root).

Summary

This paper has discussed DNS, BIND and the exploit bind8x.c. We have come up with a couple of signatures as well. First, we have the inverse query UDP packet with an ID of 'beef' and a total size of 484 bytes. Second, we have the response UDP packet with an ID of 'dead' and a total size of 538 bytes. Finally, we have a TCP destination port of 36864. By using this port the attacker will get root (if vulnerable).

Hopefully, this discussion has provided enough information to help anyone who is scratching their head about packets that are similar to the ones discussed here. If readers have further questions or concerns, below are some good related links that may provide some helpful information.

References

- [1] Stevens, W.R 1994, TCP/IP Illustrated Vol. 1. Addison-Wesley, Reading Mass p. 187
- [2] ftp://ftp.isi.edu/in-notes/rfc1035.txt p. 25
- [3] ftp://ftp.isi.edu/in-notes/rfc1035.txt p. 40

To read **Hacker Tools and their Signatures, Part Two: Juno and Unisplloit**, click [here](#).

Toby Miller is a contributing author to the book Intrusion Detection Signatures and Analysis (NEW RIDERS Publishing) and is a contributing author to Maximum Security Rev. 3 (SAMS Publishing). He has done work both in the Linux security world and Intrusion Detection/Firewall world. Mr. Miller has published numerous papers for both SANS and Securityfocus.com. He can be reached at tmiller@va.prestige.net.

Relevant Links

[DNS RFC](#)

[IETF](#)

[BIND Vulnerabilities](#)

[CERT](#)

[NAI Bulletin](#)

[PGP Security](#)

[Privacy Statement](#)

Copyright 2006, SecurityFocus