General: Why is rsh not working?

Why does RSH not work?

Symptoms

- *rsh <hostname> <command>* results in poll: protocol failure in circuit setup

- */var/log/messages* contains in.rshd message like "Could not allocate space for cmdbuf"

- *strace -f in.rshd* contains messages like:

24105 mmap(NULL, 46116860184273932000, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

24105 brk(0x40002aaaaacd1000) = 0x2aaaaacb0000

24105 mmap(NULL, 46116860184275230720, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

24105 mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_

NORESERVE, -1, 0) = 0x2aaaac765000

Cause

Looking into netkit-rsh's source code one notices the following:

(file rexecd.c)

cmdbuflen = sysconf (_SC_ARG_MAX);

General: Why is rsh not working?

```
if (!(cmdbuflen > 0)) {

syslog (LOG_ERR, "sysconf (_SC_ARG_MAX) failed");

fatal ("sysconf (_SC_ARG_MAX) failed\n");

}

...

cmdbuf = malloc (++cmdbuflen);

if (cmdbuf == NULL) {

syslog (LOG_ERR, "Could not allocate space for cmdbuf");

fatal ("Could not allocate space for cmdbuf\n");

}
```

The problem lies in the malloc statement as it is trying to allocate _SC_ARG_MAX bytes of memory. _SC_ARG_MAX represents ARG_MAX, as defined by the values returned by sysconf(), the maximum number of bytes of arguments and environment data that can be passed in an exec function. Traditionally Linux used a hardcoded: #define MAX_ARG_PAGES 32

to limit the total size of the arguments passed to the execve() (including the size of the 'environment'). That limited the maxlen of the arguments passed to about 128KB (minus the size of the 'environment'). With Linux-2.6.23, this hardcoded limit was removed. ARG_MAX is not statically defined anymore by glibc (i.e. in a header file), but glibc computes its value from the userspace stack size (sysdeps/unix/sysv/linux/sysconf.c)

The following C program prints the value of _SC_ARG_MAX when executed on a Linux system running kernel 2.6.32:

```
[root@node001 ~]# cat test.c

#include <stdio.h>

#include <unistd.h>
```

General: Why is rsh not working?

```
int main() {

    return printf("ARG_MAX: %ld\n", sysconf(_SC_ARG_MAX));

}
```

```
[root@node001 ~]# gcc test.c  -o test

[root@node001 ~]# ./test

ARG_MAX: 4611686018427387903
```

Since RSH is setting its command buffer size value to _SC_ARG_MAX , this would require a huge amount of memory to be allocated and such an amount of memory is unlikely to be present in most computers!!!

Solution

A.

- Create a wrapper around rsh that sets a reasonable userspace stack size:

```
#!/bin/bash

ulimit -s 262144 /usr/sbin/in.rshd
```

The numeric argument to the ulimit command is the number of kilobytes, so the 262144 in the example above stands for 209715200 bytes, which is exactly 256 MiB. Also, note, that 'ulimit' is a sh/bash builtin command. The number you specify when editing a file in '/etc/security/limits.d/' is also the number of kilobytes, so an example line for setting the stack size globally in this way (instead of with the wrapper) would be the following: * - stack 262144

- Edit '/etc/xinetd.d/rsh' so that it points to the wrapper instead of to the original 'in.rshd' binary by altering the 'server' directive in that file.

General: Why is rsh not working?

B. (Nov. 2013)

You can opt to install the patched rsh RPMS provided by Bright. The RPMs can be downloaded from:

http://support.brightcomputing.com/rsh/

Unique solution ID: #1153
Author: Panos Labropoulos
Last update: 2013-11-25 20:12