

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

For many, access to the router's command line interface (CLI) through telnet or Secure Shell (SSH) is more than adequate. For others, a more "machine friendly" interface is required. A more "machine friendly" interface, you ask? In past articles I have discussed router management and configuration using EXPECT-based shell scripts. In an effort to provide "fair and balanced" discussion, this month we will look at implementing IOS r-command support and an alternative to Virtual TTY (VTY) CLI access. Despite its potential security drawbacks, r-command provides a greater degree of flexibility to network managers who utilize scripting for router monitoring and management.

What Are The R-Commands?

The r-commands (the "r" stands for remote) **rlogin**, **rcp** (r-copy), and **rsh** (r-shell) were developed in the 1970s. Part of the Department of Defense funded a TCP/IP development effort at the University of California at Berkeley. R-command support was first distributed as a production service in the BSD 4.2 Unix distribution -- the first production TCP/IP networked operating system. The premise of the r-commands is to provide a suite of tools that allows a user to login, copy and execute command on remote systems without any additional authentication beyond their initial local system login. R-command "authentication" works using a "trusted peer" model. User access is defined on each host either globally using the `/etc/host.equiv` file or locally with `$HOME/.rhosts` file. The files define a combination of trusted hostnames or IP address entries with trusted usernames. Both files use the same syntax format for defining user access. Here are the syntax examples for `/etc/hosts.equiv`:

```
"+ +"
```

All hosts and all users are trusted, any user to login from any host.

```
"oedion.outland.net +" or "172.16.100.2 +"
```

The host oedion is trusted. Any user on oedion or 172.16.100.2 to connect may to the host

```
+ martin
```

The user martin is trusted and may connect to the host, from any host.

```
fury.outland.net zach
```

The user Zach is trusted only from the fury.outland.net.

In the case of `$HOME/.rhosts` there is no need for a user definition since the file is stored as a hidden file in a user's home directory. The "user" variable is derived from the owner of the `$HOME` directory. Here are syntax examples for `.rhosts`:

```
"+"
```

All hosts are trusted and the user can login from any host.

```
"oedion.outland.net +" or "172.16.100.2 +"
```

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

The host oedion is trusted and any user may login from oedion.

+ fred

All hosts are trusted and the remote user fred can login as \$HOME user.

With such a lightweight authentication scheme, great caution should be taken. After stating the obvious, keep in mind the original assumption with the r-commands is that the "peer hosts" are contained in a closed, secure environment. The r-commands were not intended for users across the Internet, but they are more than adequate for use within a secured private network environment.

R-command OS Support

R-command support is available on all Unix flavors, Mac OS X and Windows NT/2000/XP. Unix and Mac OS X systems support the full r-command complement: rlogin, rcp and rsh. Windows NT/2000/XP systems only support rcp and rsh. Here's the CLI syntax and functional description for each of the r-commands. Note there is a slight difference between the Unix and Windows syntax:

1. RLOGIN

The rlogin or remote login command opens a command shell on the remote system. On Unix systems the command syntax is:

```
user@system$: rlogin -l username host
```

or

```
user@system$: rlogin username@host
```

2. RCP

The rcp or remote copy command copies a local file to a remote host. On Unix systems the command syntax is:

```
user@system$: rcp dir/file username@host:/dir/filename
```

On windows systems the rcp command syntax is:

```
C:rcp dirfilename user hostname:/dir/filename
```

3. RSH

The rsh or remote shell executes a command on a remote system and outputs the results on the executive host. On Unix systems the command syntax is:

```
user@system$: rsh host command
```

or

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

```
user@system$: rsh -l username host command
```

On windows systems the rsh syntax is:

```
C:rsh host -l username command
```

Why Use R-Commands?

You may be asking yourself, "Why would I want to implement an inherently insecure access method, especially to manage my routers?" Well, that's a good question. The value of the r-command suite is twofold. First, there is the convenience. The r-commands were originally conceived to provide easy access to development and production systems, without the added hassle of having to authenticate each time access was required. In a router environment, the r-commands make it easy to quickly see critical status information.

The second, and perhaps more persuasive benefit is flexibility. Because remote command output is displayed locally, it can be manipulated locally. This is a big win for scripting and developing monitoring tools. Want to parse the routing table with AWK, use rsh to <show ip route> and pipe (the Unix "|", for redirecting stdIO) the output into a file or directly into AWK. The r-command approach provides a great deal more data processing options than using EXPECT "fetch and save" based scripts, because the data can be handled directly, instead of being collected, then post-processed. Now let's look at how to configure the router to support r-command access.

Configuring IOS R-Command Support

Implementing IOS r-command support is very simple process:

1. Enable r-services
2. Configure Host/User Trust
3. Configure Router Access Security

Enable R-Services

The IOS supports inbound and outbound (somewhat, depending on the upstream server and the router's hostname length) rcp and rsh access. Rlogin is not supported, except via Kerberos. RSH and RCP server support must be enabled individually. This provides some flexibility as far as the degree of r-command access you wish to make available. For example, if you want to copy files to the router's file system or the startup or running configuration files, you would enable only the rcp service. Here is a configuration syntax example enabling both rcp <ip rcmd rcp-enable> and rsh <ip rcmd rsh-enable> services:

```
r-3640(config)# ip rcmd rcp-enable
r-3640(config)# ip rcmd rsh-enable
r-3640(config)# ip rcmd source-interface Ethernet 0/0
r-3640(config)# ip rcmd remote-username cisco-2600
```

The <ip rcmd source-interface {interface}> command sets the source IP address of the outbound r-command requests. This is the IP address that is added to the /etc/hosts.equiv file of hosts to which the router is permitted to execute rsh requests and copy files via rcp. The username that is associated

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

in the /etc/hosts.equiv file with the router's IP address is the router's hostname. This is the local user that is executing the rsh/rcp request. The <ip rcmd remote-username {username}> defines the username on the remote host where rsh and rcp requests are executed. This account must exist on the trusted host.

Configure Host/User Trust

Once inbound rcp and rsh support has been enabled, the host/user trust equivalencies must be configured before any user can access the router using rcp or rsh. Trust equivalences are configured using the configuration command <ip rcmd remote-host {local username} {host list ACL # | hostname/IP add} {remote username} [enable (1-15)]>. Each <ip rcmd remote-host> command entry is the IOS equivalent to a /etc/hosts.equiv entry on a Unix system. It's this local data set that is used to authenticate inbound r-command requests. Let's look at a command syntax example, then define the different values:

```
r-3640(config)# ip rcmd remote-host martin 22 martin enable
r-3640(config)# ip rcmd remote-host blob 172.30.71.5 martin enable
```

The first example uses a standard access list to define the trusted hosts from which the trusted local/remote user "martin" can run rcp and rsh commands. The second example says the trusted local user "blob" can execute rcp and rsh commands on the router from the trusted host 172.30.71.1 logged in as martin (remote user). Got that? If you did not, that's OK. Because while the r-commands have no password security, they do require that the host local and remote host trust definitions exist, if they don't there is no access. To get a better understanding, let's look at how an rsh command is authenticated. On our Unix host, using the second syntax example (local user blob and remote user martin) we run the command:

```
172.30.71.5$ rsh -l blob r-3640 show running-config
```

The router sees the request as remote user martin and executes the command "show running-config" as the local user blob. It refers to its "remote-host" database and verifies the trust:

```
ip rcmd remote-host blob 172.30.71.5 martin enable
```

Then the rsh command request is executed. User authentication is dependent on the remote user and host values; in this case it's the IP address 172.30.71.5 and the username martin. The output of the command is dictated by the local user value blob, which in this case has 15 privilege-level access. The same holds true for rsh commands run from the router.

When this is the case, the router's local user name is the router's hostname and the host address is the IP address defined as the <ip rcmd source-interface {interface}>. On the router we execute the command:

```
r-3640#rsh trinity /user martin ps -aux
```

The Unix host interprets the request that the remote user r-3640 is executing a command from the

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

host 172.39.66.1 as the local user martin. The Unix host trinity checks its /etc/hosts.equiv file and finds the entry:

```
! hosts.equiv Trinity 172.30.71.5 C Date 12-19-98
172.39.66.1 r-3640
```

It also finds the local user "martin":

```
martin:*:0:100:Michael Martin:/home/martin:/bin/bash
```

With the trust credentials found, the system executes the command. While this is not overly complicated, it is helpful to understand the authentication mechanics when creating the trust relation definitions. Remote r-command execution is not a predicate for rsh/rcp access to the router, but it is a requirement to support remote file system copy from the router to another host via rcp.

Configuring Router Access Security

The final step is limiting the hosts with which the router can interact using the r-commands. To control inbound r-command access, use the VTY access control mechanism <access-class>. An access-class statement applies inbound (and outbound) host access control to the VTYS, restricting ip rcmd, telnet, and SSH access to the hosts listed in the standard IP access list. To start, we define a standard IP access list:

```
r-3640(config)#access-list 4 permit host 172.30.71.4
r-3640(config)#access-list 4 permit host 172.30.71.6
r-3640(config)#access-list 4 permit host 172.30.71.48
```

Then we drop in to line VTY configuration mode, define which VTYS we want to modify, define the protocols accepted on those VTY, and then apply the access group:

```
r-3640(config)#line vty 0 4
r-3640(config-line)#transport input rlogin telnet
r-3640(config-line)#access-class 4 in
```

We're done. Only the hosts defined in access list 4 can access the host via telnet, SSH, or the r-commands. If you wish to restrict the hosts with which the router can interact via outbound r-commands, an outbound IP access group must be applied to the interface defined as the <ip rcmd source-interface>. Here is an extended access-list example that permits ip rcmd between one host and permits all other IP services.

```
access-list 120 permit tcp host 172.30.71.5 host 172.30.71.100 eq cmd
access-list 120 permit tcp host 172.30.71.5 host 172.30.71.100 established
access-list 120 deny tcp any any eq cmd
access-list 120 permit ip any any
```

That concludes our look at implementing r-command support on IOS-based routers. I hope that you found this installment useful. I think this is really a really cool IOS option, even if it has some potential

More Cool IOS Commands - Implementing r-Command Support

Michael J. Martin

security implications. If implemented properly and in the right environment, it can be utilized to great advantage. As always, questions and comments are welcome, especially article ideas.