ORNL-RSH package and Windows '03 PVM 3.4

Phil Pfeiffer[1], Stephen L. Scott[2], and Hardik Shukla[2]

[1] East Tennessee State University, Dept. of Computer and Information Sciences,
Box 70711, Johnson City, TN 37614-1266
phil@etsu.edu

[2] Oak Ridge National Laboratory, Computer Science & Mathematics Division,
Bethel Valley Road, Oak Ridge, TN 37831-3637
scottsl@ornl.gov hs2@ornl.gov

1. Introduction

The first public release of PVM was version 2.0 in February 1991; the first PVM release from Oak Ridge National Laboratory that supported the Windows operating system was version 3.4 (beta 5) at the end of 1997 and then with the formal release of PVM version 3.4.0 in January 1999. While this initial release provided the PVM framework and functionality to the Windows world, there were a number of shortcomings that created undue difficulties for users. Some of the problems encountered included the difficulty of the installation process itself, the necessity for a 3rd party and costly rsh package, and the lack of Windows user level documentation of the entire process (from install to use). Thus, the general goal of this work and its resulting release is to simplify the use of Windows PVM by its user community. To achieve this goal, the new package will have to include: a simplified installation process on individual and networked machines (clusters), a simplified and free (open source preferred) rsh package for PVM, and of course the associated documentation so that others may more easily use all the new software. This paper contains a brief introduction of new features included in the latest Windows PVM release and spends the remainder of the paper discussing the details of the new ORNL-RSH package that greatly simplifies the use PVM on Windows based machines.

2. Windows PVM v3.4

Windows '03 PVM v3.4 differs from earlier versions of Windows PVM in three key ways:

- '03 supports Windows 2000 and Windows XP (new), as well as Windows 98 and Windows NT.

- ’03 includes ORNL-rsh, an open-source rsh daemon for Windows platforms (new). ORNL-RSH, which was created expressly for Windows PVM v3.4, is described in more detail in Section 3.
- ’03 supports a simplified installation procedure, as follows:
  - Support for client-mode installation, a subset of the standard PVM package designed for hosts that only issue commands, was dropped, to make the PVM installation dialogue less confusing to novice PVM users.
  - Support for remote installation was enhanced, with the aid of a new PVM installer that fully supports remote PVM installation. Prior to v3.4, anyone who installed Windows PVM on a host H had to make hands-on contact with H to complete the installation. Even when a network share was used to distribute PVM to host H, the administrator was required to physically visit H, and run a script locally on H that downloaded PVM to H, then configured PVM for local use.

To use the PVM installer, a user must have an administrator-level account on every host involved in the PVM installation procedure—local and remote hosts alike. Any remote account that the administrator intends to use must also be network accessible. Under Windows XP, network access for administrator accounts must be explicitly enabled using XP’s security settings.

The new PVM installer’s look and feel resembles that of the earlier, InstallShield™ based installer. A new screen for configuring remote installation supports the use of wildcards to target sets of hosts and IP addresses for PVM installation. Other new screens confirm a configuration’s parameters before starting an install, and confirm daemon installation.

3. ORNL-RSH

Windows PVM uses the *rsh* protocol to execute commands on remote machines: e.g., to start a PVM daemon on a remote host. In rsh, a string like

```
rsh -l remote-user remote-host hostname
```

represents a request to run a command *hostname* on a host *remote-host* as user *remote-user*. A request is typically accepted by a program known as an *rsh client*, which relays it to the specified host for further processing. This processing is done by programs known as *rsh servers*, or *rsh daemons*. Both clients and servers are needed for full rsh operation: a host that lacks rsh client code cannot initiate remote commands, and a host that lacks an rsh daemon cannot respond to requests from remote hosts.

Heretofore, the use of PVM under Windows has been impeded by a lack of public domain rsh servers for the Windows environment. rsh servers, which were originally developed under BSD Unix, do not port readily to the Windows environment, thanks to important differences between the Unix and Windows models for user logins, tasking, and

security. Commercially available implementations of rsh servers for Windows PVM currently cost as much as $120 (USD) per machine to license.

The ORNL-RSH software package, which is installed as part of Windows PVM v3.4, is a freeware product, distributed under an open source license similar to PVM's [1]. This package, which can be obtained from [1], includes an rsh daemon for Microsoft's Win32 subsystem, and supporting utilities that install this daemon on local and remote hosts; manage its operation on remote hosts; and uninstall the package's utilities and files.

The ORNL-rsh daemon is installed as a Win32 service that runs under a Windows administrator account. A supporting database holds Windows-specific process execution parameters—information that cannot be obtained via the basic rsh protocol. This database also allows administrators to restrict every remote user's access to a prespecified set of local accounts.

The balance of this section discusses ORNL-RSH's features, supporting tools, and security limitations in more detail. The section, which assumes a prior familiarity with rsh, focuses primarily on extensions and accommodations for the Windows environment.

3.1 ORNL-RSH Features

The ORNL and BSD rsh daemons implement similar protocols for processing user requests. For example, executing a command like

```
rsh  -l  someuser  someWindowsHost  hostname
```

on an ORNL-rsh daemon returns the name of the remote host. Executing a command like

```
rsh  -l  someuser  someWindowsHost  cmd.exe
```

has the effect of opening an interactive session on the remote host.

The two daemons also differ in four important regards. ORNL-rsh eliminates interactive challenges for user passwords; limits remote access to prespecified users; supports Windows-specific options for managing command execution; and supports syntax extensions for the Windows environment.

No password challenge. The BSD daemon, by default, uses an interactive password challenge to authenticate requests for service. Depending on how an rsh client is implemented, this challenge is made visible to the remote user—

```
user@host> rsh -l targetacct targethost hostname
password:         # challenge returned by targethost login
```

—or managed transparently, with an additional command-line parameter—

```
user@host> rsh -l targetacct -p passwd targethost hostname
```

Neither style of challenge is secure: rsh fails to encrypt passwords in transit, thereby rendering them vulnerable to scanning [2]. ORNL-RSH currently omits this interactive

password challenge, along with two BSD rsh mechanisms for challenge bypass: *.rhosts* and *hosts.allow* files.

Explicit limits on remote access. BSD rsh uses "implicit allow/explicit deny" to manage access to a server host. BSD's rsh daemon vets any request from any host not named in *hosts.deny*, an auxiliary file of untrusted hosts, regardless of the account from which the request came, or the account that it targets.

ORNL-rsh uses an "explicit allow/implicit deny" policy to manage access. The daemon *accepts* a request of the form

```
user@host> rsh -l targetacct targethost hostname
```

if and only if *targetacct* is registered in an ORNL auxiliary database of *rsh* accounts; *someuser@somehost* matches an entry in the list of *targetacct's* users; and the request originates from *somehost*—as determined by a check of the logical connection between *somehost* and *targethost*.

Support for Win32-style command execution. The ORNL and BSD rsh daemons respond to a request to run a command as *targetacct* by creating a new process that runs as *targetacct*, and then using this process to run that command. The ORNL mechanism for command execution, however, is more complex than the corresponding Unix mechanism, thanks to three key differences between the Unix and Win32 environments.

Difference 1: domains. The standard UNIX login mechanism presents a one-level view of the network environment, associating every computer with a unique set of accounts. The Win32 subsystem presents users with a *two*-level view of the networked environment. Every computer in a Win32 environment is associated with one *or more* uniquely named views of the network, known as *domains or workgroups*, that capture distinct views of a network's directories, devices, and servers. Every workgroup or domain, in turn, is associated with its own set of accounts.

Under the Windows model of system access, a user, at time of login, can access one of several accounts that bear his name: *e.g.*, an account named *phil* in a domain specific to the local host, or an account named *phil* in a domain specific to the larger network. Accordingly, Win32 logins require three, rather than two, parameters: an account, a password, *and* a domain. This third, domain parameter is sometimes hidden from the Windows user with a special login screen that automatically supplies a default domain—but such screens simply mask the need for the domain parameter, and do not eliminate it.

Difference 2: stricter security. UNIX allows root users to configure daemons can "impersonate" any user ***U*** at any time. The Win32 model of process security allows impersonation only when the daemon has explicit permission to become user ***U*** (cf. [3]). This permission can be obtained in one of two ways:

- *Using* named pipe *authentication*. A daemon can use *named pipe authentication* to get permission to impersonate user ***U***. Permission must be obtained from a task that is already running as ***U***; that has *explicit* permission to grant other tasks the right to impersonate ***U***; and that is running in a domain that the daemon's host domain trusts.
- *Using passwords*. A daemon may also obtain permission to impersonate ***U*** by supplying the operating system with ***U***'s password.

Difference 3: No home directories. Every Unix account is associated with a home directory, which the BSD rsh daemon uses as an initial directory for running commands. Windows accounts have no default home directories, which makes the BSD strategy for determining an initial directory unworkable.

These three differences create a need for three new task-creation parameters in ORNL-rsh: the name of a default domain to access when logging into an account; the local account's password, which is needed to support the impersonation of local users in a non-interactive way; and the name of a directory to be used as a local working directory. The ORNL-rsh daemon obtains these three parameters from its auxiliary database, which must be preconfigured before using ORNL-rsh.

Extended command line syntax. ORNL-rsh supports the use of Unix-style and Windows-style command-line syntax.

- ORNL-RSH recognizes Unix- and Windows-style environment variables—i.e., $THIS and %THIS%—and expands variables in the context of the current user's environment.
- ORNL-rsh recognizes Unix- and Windows-style pathname separators—e.g., treating, "C:/WinNT" and "C:\WinNT" as synonyms for the standard Win32 master directory.
- ORNL-RSH recognizes UNC-style syntax for account names—e.g., treating a reference to user \\DOMAIN\USER as a reference to an account named USER in a domain named DOMAIN. Any account name that is not prefixed with \\DOMAIN\ is treated as a request that specifies the default local domain, \\.

3.2 ORNL-RSH Tools

The ORNL-rsh daemon is distributed with four supporting utilities: an installer utility that installs the daemon on local and remote hosts; a configuration tool that configures the rsh daemon for use on a local host; a manager tool that exports a local rsh configuration to a set of remote hosts; and an uninstaller.

ORNL-RSH Installer. The ORNL-RSH distribution consists of a single, self-installing executable. This program installs the ORNL-RSH package in four phases:

1. The administrator first configures ORNL-RSH for the target environment. The administrator selects the directory in which to install the RSH package; specifies the account under which the rsh daemon will run; and specifies what files, if any, will be installed remotely.
2. The administrator specifies what remote hosts, if any, to target for package installation.
3. The installer utility installs the specified pieces of the ORNL-RSH package on the specified hosts, displaying a running update of the package's installation status.
4. The installer gives the administrator the option of restarting the targeted hosts, for the purpose of activating the ORNL-RSH service.

ORNL-RSH Configuration Tool. After ORNL-RSH has been installed, the rsh daemon must be configured for every host on which it resides. The ORNL-RSH configuration tool allows an administrator to configure a daemon for use on the *local* host—the host on which the configuration tool is running. The tool's options allow the administrator to

- configure an account for use as an *rsh account*—i.e., a local account that remote rsh users can use to run commands. This task requires the administrator to specify the account's domain and password, as well as an initial directory for running commands.
- associate an rsh account with remote users—i.e., users authorized to use that account to run their commands.
- list all remote users that are associated with a particular rsh account.
- end an association between an rsh account A and a remote user.
- remove an account from the list of rsh accounts.

ORNL-RSH Manager. The ORNL-RSH manager utility allows the administrator to manage *remotely* installed copies of ORNL-RSH. The utility supports options that install ORNL-RSH on an administrator-specified set of hosts; export a locally configured rsh database to these hosts; start and stop remote ORNL-RSH services; and uninstall ORNL-RSH remotely.

ORNL-RSH Uninstaller. The uninstaller removes ORNL-RSH and its supporting databases from the specified hosts, deactivating rsh daemons in the process.

3.3 ORNL-RSH Security Limitations

Authentication based on point of origin. Shortly after it was developed, the rsh password challenge protocol, which transmits passwords in clear text, was rendered insecure by devices like protocol analyzers. The strategy for remote user authentication described here "solves" the clear text password transmission problem by authenticating requests based on their points of origin. Unfortunately, accepting requests based on point

of origin allows anyone who can compromise a remote account, or spoof their point of origin with techniques like DNS poisoning [2], to run commands on a rsh server.

A better strategy for secure authentication uses encrypted channels to protect user passwords. The main problem with such protocols, from the standpoint of PVM, is a lack of backward compatibility with existing implementations of rsh.

Stored passwords. Section 3.1 noted that Win32's non-support for unrestricted impersonation forces any third-party service that acts on behalf of a user ***U***, in the worst case, to supply the Win32 subsystem with ***U***'s password. Moreover, if this service is to run without direct user interaction, passwords must be stored on electronic media—thereby rendering passwords vulnerable to compromise.

ORNL-RSH stores the password for every local rsh account in its host system's registry. All of these passwords are encrypted using Blowfish [4], with a single master key. This key is stored in a second registry key, *HKEY_LOCAL_MACHINE\Software\Rshd*, in a field named *Rshd_Key*. Access to this registry key is limited to users with administrator privileges. The master key is generated once, at random, when the password database is generated, and used for all subsequent password encryptions.

One serious problem with storing any password on any system is that the systems themselves are vulnerable to hacking. For example, Scambray et. al. describe a variety of strategies for obtaining unauthorized access to protected data in the Windows registry [5]. A typical hack involves first dumping the registry, using security holes like improperly secured network connections, improperly secured bootstrap procedures, and registry backup utilities, and then analyzing the dump on a second, hacker-friendly system. The registry can be made more secure, using strategies like disallowing remote registry access (which would render ORNL-RSH inoperable); password-protecting BIOSes; and registry encryption (cf. [6]): even so, the password remains on the system.

An alternative approach to protecting the master key would be to store this key outside the registry—for example, in a main memory location whose contents are generated at runtime, using a "secret" key generation algorithm. The authors' desire to make ORNL-RSH an open source utility precludes the use of secret algorithms to protect keys. In any case, a blind reliance on obfuscation to secure data is a practice that is held in low esteem by most of the security community, particularly in light of unsuccessful attempts to use "secret" approaches to protect data (e.g., DIVX, DeCSS).

4. Conclusion

As stated at the beginning of this paper, the general goal of this work and its resulting release is to simplify the use of Windows PVM for the user community. Due to the page limitation we were only able to provide significant details regarding the ORNL-rsh package and much of that is at the design and administrator level. Comprehensive user level documentation is undergoing an external beta test and review process as of this

writing and will be released on the ORNL PVM web site [1] prior to the EuroPVM/MPI 2003 meeting in September.

Regarding our future plans for Windows development, one significant goal for the development team is to leverage the experience gained in this effort to enable us to expand the current state of Windows based high-performance computing and cluster computing to a more Windows centric manner while: still supporting native HPC codes written in their original UNIX/Linux format, transitioning new codes to be written for Windows HPC in a Windows centric manner, and enabling both the aforementioned items to coexist. While some have proclaimed Windows HPC as dead, we have seen a recent resurgence of interest in cluster computing using the Windows computing environment. In some cases, a business is looking to leverage Windows desktops during non-peak utilization. In other instances it is a new type of user that has never even considered traditional HPC but is instead looking for a way to leverage cluster and HPC characteristics in manners not traditionally considered in either of these two camps.

Regardless of your use, we hope that this will serve as an impetus for you to further investigate the new Windows PVM release, ORNL-rsh, and associated packages.

References

[1] Oak Ridge National Laboratories, "PVM: Parallel Virtual Machine", www.csm.ornl.gov/pvm/, (last accessed April 5, 2003).

[2] W. Cheswick and S. Bellovin, Firewalls and Internet Security: Repelling the Wily Hacker, © 1994, Addison-Wesley.

[3] D. Solomon and M. Russinovich, Inside Microsoft Windows 2000 (Microsoft Programming Series), © 2000, Microsoft Press.

[4] B. Schneier, Applied Cryptography, 2nd Edition, c. 1996, John Wiley and Sons

[5] S. McClure, J. Scambray; and G. Kurtz, Hacking Exposed: Network Security Secrets & Solutions, Third Edition, © 2001, McGraw-Hill Osborne.

[6] M. Edwards and D. LeBlanc, "Where NT Stores Passwords", *Windows & .NET Magazine*, Aug. 1999 [www.winnetmag.com/Articles/Index.cfm?ArticleID=5705], (last accessed January 29, 2003).

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.