Global Information Assurance Certification Paper

A Case Study

Removing Server Based Trust Relationships

GIAC-Security Essentials Certification (GSEC)
Practical Assignment Version 1.4b
Option 2

Keith B. Gaughan
February 23, 2004

Table of Contents

Introduction

The goal of this project was to develop, implement and deploy solutions as well as supporting processes and standards to remediate and mitigate the risks that are inherent to utilizing UNIX server based trust relationships in a enterprise networked environment within 30 days. Server based trust relationships can be defined to grant different levels of authenticated or unauthenticated access. Trust between hosts can be established at the user level or globally at the server level. Corrective action is required to remediate and or mitigate the risks created by the currently established trust based relationships and the lack of controls to prevent their establishment. In addition to the corrective actions, alternative approaches and solutions that have a greater emphasis on security and access accountability will be introduced and implemented.

The completion of an internal audit assessment of the UNIX platform, with an emphasis on the Solaris operating system, revealed excessive use of configuration files that establish and govern trust based relationships between servers within the corporate production network. The findings specifically targeted root level .rhosts and /etc/hosts.equiv files. In response to the internal assessment the UNIX Engineering Services (UES) team, of which I was a member, and the Data Security (DS) team successfully identified corporate wide trusted hosts, assessed the level of risk posed by existing relationships, developed and implemented corrective actions that centered around the use of Secure Shell (SSH), established compliancy monitoring of deployed solutions and development of documented standards.

Original State of the Environment: *The Before*

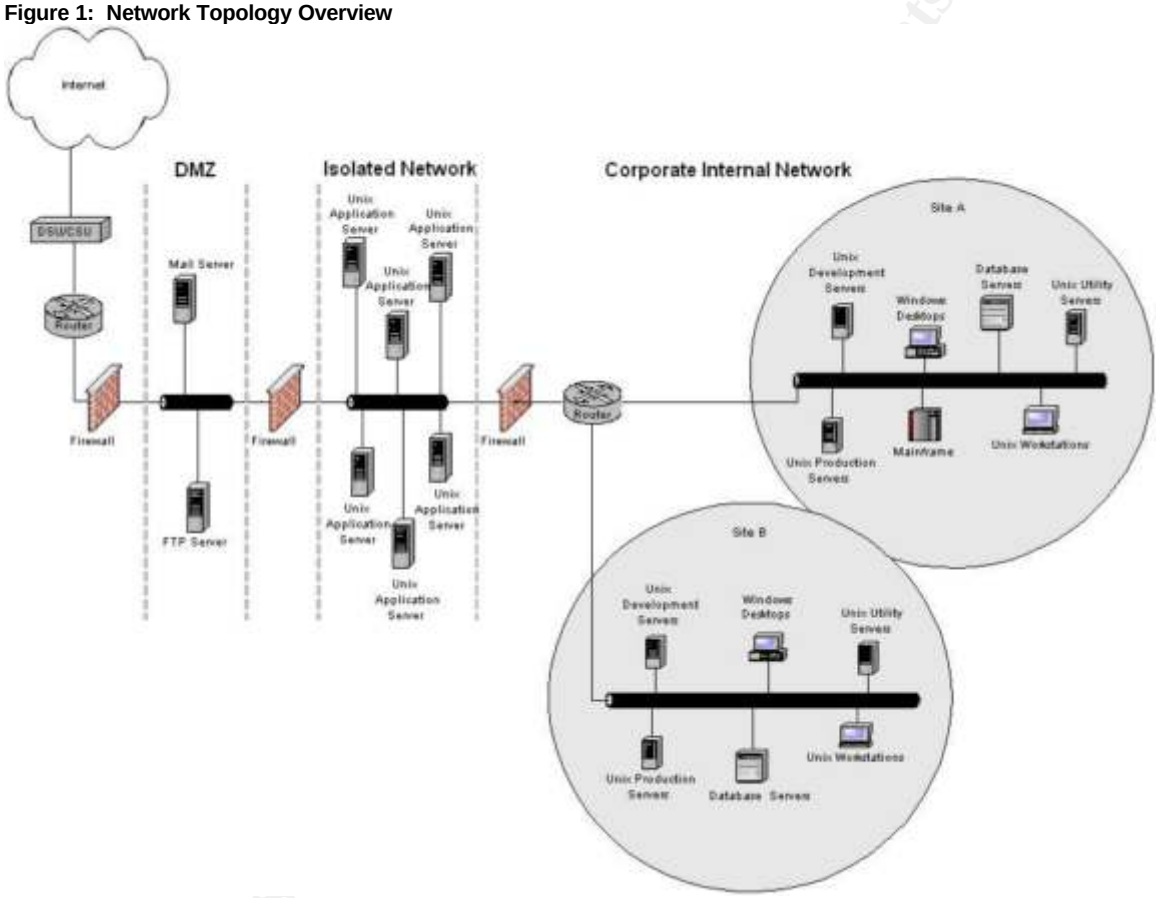
Environment and Infrastructure Overview

The current networked environment consists of servers and network devices that are typical of a large diverse organization. Technologies found throughout the enterprise include such platforms as Solaris, AIX, Linux, Windows and Mainframes in addition to a variety of network devices such as but not limited to routers and switches. Our efforts were targeted directly to the Solaris platform. The Solaris server base consisted of approximately 450 servers of various hardware architectures ranging from Enterprise 12Ks to Sparc 20s and accounted for over 90 percent of all deployed UNIX servers within the enterprise.

The currently implemented network topology can be referred to as flat in nature with minimal separation between network segments. In general, the network security architecture is poor because development and production systems are located on the same networks. Additionally, management, operational and end-user traffic flow over the same networks as production data. A generic depiction

of the current network topology is illustrated in *figure 1*. The illustration shows that the only defined separation occurs in the demilitarized zone (DMZ), which supports a few servers for the purpose of email, external ftp and the isolated network where application servers provide services to external customers. The bulk of the core infrastructure resides in what is referred to as the CIN (Corporate Internal Network). Internal applications such as human resource systems, corporate finance systems, customer applications and general end-user traffic all reside within the CIN portion the network.

Figure 1: Network Topology Overview



The current network topology greatly enhances the risks that result from the establishment and use of trust relationships because development and production servers, administrator workstations, end-user desktops and even servers within isolated networks can be setup to trust one another. The lack of segmentation allows unimpeded access to servers in both isolated networks and the CIN portion of the network. In *Figure 2*, a more detailed version of the CIN an isolated portion of the network as shown in *figure1*, you will see examples of typical trust relationships that are a common occurrence between servers, workstations and desktops.

Figure 2: Sample Trust Relationships

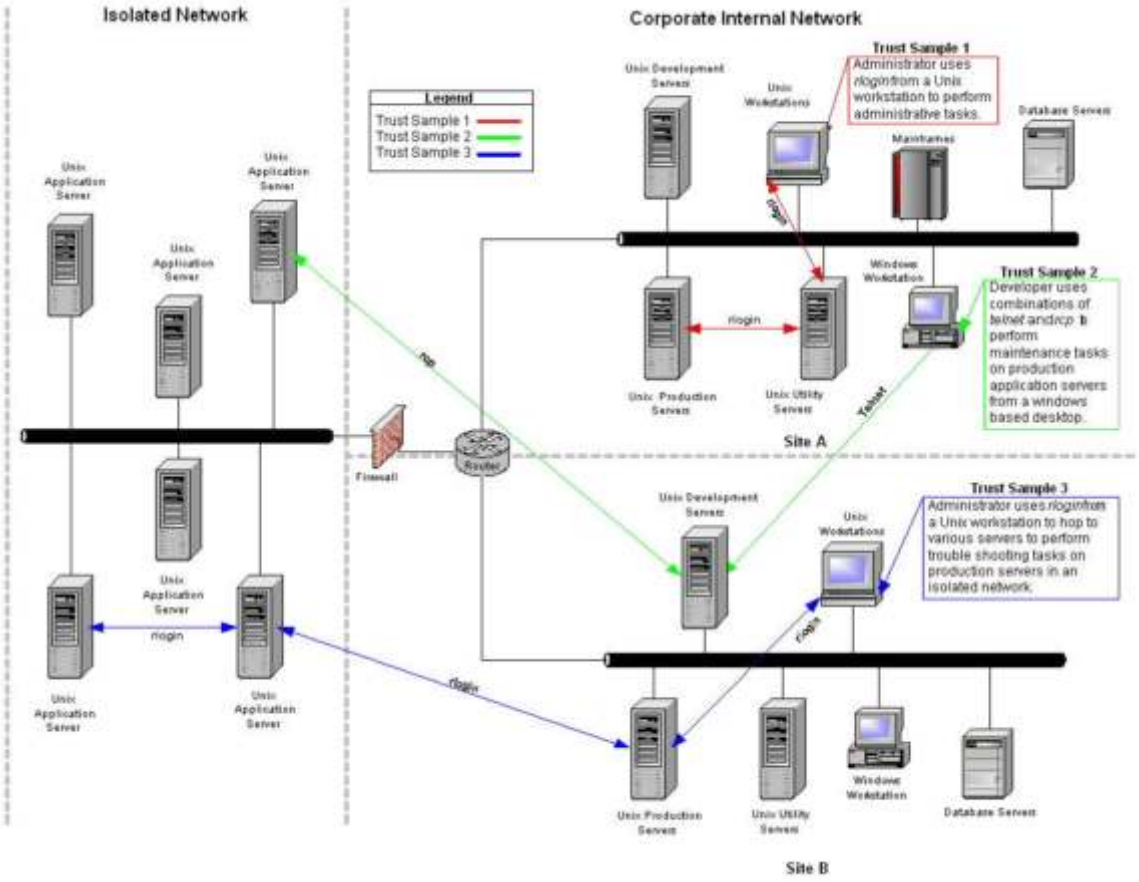


Figure 2 highlights three separate examples of trusted relationships. The first example “Trust Sample 1” (marked in red) highlights a system administrator as he logs onto a utility server from his workstation using rlogin, then hops to the trusted production server using rlogin again. This process would be a very common occurrence because administrators would require local accounts on many servers and using rlogin provides the convenience of not having to continually enter username and passwords. The second example “Trust Sample 2” (marked in green) illustrates a developer from a windows based desktop performing a telnet session to a development UNIX server. From the development server, an rcp (remote copy) is done to copy data files to a trusted production server located in the isolated network. This type of activity is common because developers do not typically have direct access to production serves. In this instance, the developers would telnet to the development UNIX server then assume the identity of an application user account that would be trusted by the production server in the isolated network. Finally, the third example “Trust Sample 3” (marked in blue) highlights a systems administrator as he logs onto a production server from his workstation using rlogin and performs a series of hops to a trusted production server in an isolated network. The third example is very similar to the first example except that it shows that trusted relationships are being established between servers, even in isolated networks. The problem with the approach in example 3 is that trust relationships are being established in a

network that is supposed to have greater restrictions than the CIN portion of the network. If a host in the isolated network were to be compromised, the established trust relationships could easily allow unauthorized access to hosts within the CIN portion of the network.

Problem Identification

As previously identified and determined by the internal audit team, excessive use of server based trust relationships have introduced an unnecessary level of risk that requires corrective action. According to ITU-T X.509, Section 3.3.54 trust is defined as follows: "Generally an entity can be said to trust a second entity when the first entity makes the assumption that the second entity will behave exactly as the first entity expects" (Andert, p2).

A trusted relationship is created when the normal standard password-based user authentication mechanism is bypassed. A trusting server will allow users to access or run commands on the local server from a remote host without having to supply a password (Cole, p41). The remote authentication procedure determines whether a user from a remote host should be allowed to access the local system with the identity of a local user. Authentication is then granted or denied based upon policies explicitly set to allow users to use remote privileges without verifying their credentials (password).

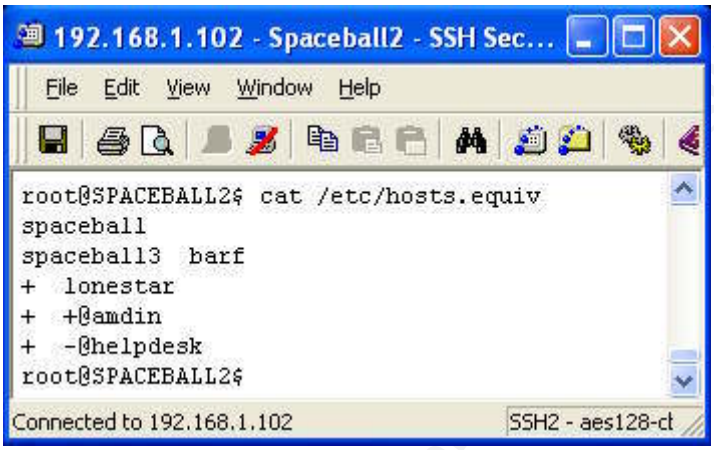
Within the UNIX platform there exists a set of commands know as the "r-commands". The name for these commands is derived from the fact that each of the commands begins with the letter "r". The list of commands is rcp (remote copy), rsh (remote shell), rexec (remote execute), rdist (remote distribution) and rlogin (remote login) (Gregory, p169-170). Trust for a server, using the "r-commands", is controlled by two files, the /etc/hosts.equiv and $HOME/.rhosts files. The /etc/hosts.equiv file controls trust behavior on a global or system level and may be superseded by the existence of a .rhosts file in the users home directory. Entries in these files are of two forms, positive entries grant access while negative ones deny access. When a user first tries to connect to a trusted host, using one of the "r-commands", the "r-command" checks the /etc/hosts.equiv file first then the local users .rhosts file, the exception being the root account which only checks the root .rhosts file (Cole, p41). Authentication succeeds when a matching positive entry is found. Authentication fails when the first negative entry is found or if no matching entries are found in either file.

File Configurations

The .rhosts and /etc/hosts.equiv files are formatted as a list of one-line entries. Negative entries are differentiated from positive entries by a "-" character preceding either the hostname or username field. The use of the special character plus "+" can be used in place of either the hostname or username fields or both. The "+" entry acts a wild card character and would match any known hosts or users. In general, the format for these files is:

Syntax: ***hostname*** *[username]*

While the .rhosts and hosts.equiv files have the same file format, the same entries in each file have different effects. The following sample of a hosts.equiv file, from the host *spaceball2*, will be used to demonstrate some typical entries and their impact to a system. In addition, the sample will be further explained to illustrate the differences between hosts.equiv and .rhosts files. (Authentication for Remote Logins, docs.sun.com):



- A single entry for a host in the hosts.equiv file means that users from that host are considered trusted and will be granted access to the system with the same user name that they have on the remote system. In our example, the first statement creates trust for all users with local accounts coming from the remote host *spaceball*. If the user *smith* was doing an rlogin from the remote host *spaceball* and *smith* had a local account on the target host (spaceball2), that user would be granted access without supplying a password.

  The syntax and functionality for single host entries are the same for both hosts.equiv and .rhosts files.

- If the user name is also specified along with the host name then the host is trusted only for the specified user. However, if the hostname/user format is used in the hosts.equiv file, the specified remote user will be allowed access as any local user on the system. In our example, the second statement creates trust for the user *barf* from the host *spaceball3*. In this example the user *barf* would be granted access to the target host (spaceball2) as any local user, including such accounts as *sys*, *nobody* and *lp*.

  The hostname/user format can also be used in .rhosts files. Unlike the hosts.equiv file, specifying a user name in a .rhosts file only allows the specified remote user access to the target system as the local user who has the entry in the .rhosts file. For example, if the user *barf* had a local .rhosts file in their home directory with an entry of “spaceball lonestar”, this would allow the user *lonestar* from the host *spaceball* access to the *barf* account.

- The use of a plus "+" character can be used in place of the hostname and will act as a wild card that will match any known host. The third statement in our example uses the "+" character to allow the specified user *lonestar* access to the target host (spaceball2) as any local user and from any hosts.

  The syntax and functionality for using the "+" entries are the same for both hosts.equiv and .rhosts files.

- If a netgroup name is preceded by a plus sign (+) then all the hosts or users in that netgroup are considered trusted. If the netgroup name is preceded by a minus sign (–) then all hosts or users in that netgroup are considered not to be trusted. In the four statement of our example, trust is established for all hosts and all users who are a member of the *admin* netgroup. In the fifth statement of our example, all users of the netgroup *helpdesk* would be denied access from all hosts.

  The syntax and functionality for using the netgroup entries are the same for both hosts.equiv and .rhosts files with one exception. Negative entries only apply to hosts.equiv files and may be superseded by .rhosts entries in users home directories.

The above example demonstrates how individual statements can be configured to grant or deny access and the impact they have on a system. However, it is also important to consider the effect that the interaction of these statements can have on systems as a whole. The reality that multiple statements will exist in a single hosts.equiv file enforces the need to fully understand the interaction of these statements and the order in which they should be placed within the file. The order of the entries in the hosts.equiv file is important especially when both positive and negative entries exist. The search order is most critical when using the username field to specify one particular user, groups of users and especially when using the "+" sign (hosts.equiv(4), docs.sun.com). To further illustrate, take our original example of an /etc/hosts.equiv file and look at the second entry that reads "spaceball3 barf". As we discussed earlier, this example allows the user *barf* to login as any local user to the host *spaceball2* without out supplying a password. However, say the user *barf* is member of the *helpdesk* netgroup. As seen in our example, the last statement "+ -@helpdesk" should deny access from all hosts to all users that are a member of the *helpdesk* netgroup. However, due to the order of which the statements are found in the hosts.equiv file the user *barf* will not be restricted because a positive match for the user *barf* would occur before the entry denying access.

File Permissions and Ownership

Appropriate file permissions and ownership must be enforced on root .rhosts, /etc/hosts.equiv and .rhosts files in users home directories ($HOME/.rhosts). If appropriate permissions are not enforced it could allow unauthorized users to

view or modify these files to grant levels of unauthorized access. Permissions of root level .rhosts and /etc/hosts.equiv files are especially critical because of their ability to effect the system globally and grant the ability to gain access to the root account. Additionally, if a system were compromised, information contained within in these file would provide helpful information to an attacker about other systems and users that are trusted within the network. Ensuring correct file permissions and file ownership of root .rhosts and /etc/hosts.equiv files would at least ensure that an attacker would have to gain root level access to view or make modifications to these files. For instance, the following example would not be an uncommon set of permissions for the /etc/hosts.equiv file. However, having the ability view the contents of this file by non-root users could reveal trusted user and system information to an attacker.

Example: `-rw-r--r--  1 root other  64 Feb  7 15:22 hosts.equiv`

Permissions that would be better suited to ensure that root .rhosts and /etc/hosts.equiv files are not viewable by non-root users would be to ensure that the file is owned by the user root and set to read-only as in the following example.

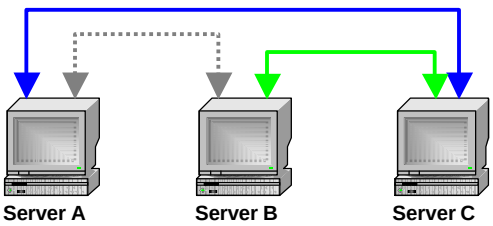
Example: `-r--------  1 root other  64 Feb  7 15:22 hosts.equiv`

File permissions of .rhosts files in users home directories are equally important in order for "-r-commands", like rlogin, to work correctly. As a feature, the user who is attempting to perform the rlogin must own the .rhosts file. For example, if the user *smith* wanted to maintain a .rhosts file in his home directory that file would have to be owned by the user *smith*.

Auditing and Validating Established Trust

The problem with establishing trusted systems is that trusting another system has implications beyond the interactions between the local system and the trusted remote system. Inevitably, trust operates in a transitive fashion and auditing or validating the relationships that form between systems becomes difficult. The following example illustrates how trust is transitive.

Example: **Trust is Transitive**
If the administrator of server A trusts server C, and the administrator for server B trusts server C, then it is a reasonable assumption that server A trusts server B.



It is critical to fully understand the extent to which trust has been granted both intentionally or unintentionally. Using the above example, the administrator of server A did not directly extend trust to server B. However, trust has been established indirectly because server B trust server C.

Identified Vulnerabilities

The establishing of trust-based relationships between servers should be discouraged and the "r-commands" that support them prohibited. The use of the "r-commands" should be prevented because of the weak authentication mechanisms that they employ. The weak authentication mechanisms allow users the ability to gain access or run commands on remote servers without having to supply passwords. In addition, even if all precautions and hardening techniques are applied, meaning users are forced to supply passwords, the "r-commands" them selves still transmit data via clear text protocols. The use of clear text protocols makes the data transmitted between the user and the server vulnerable to any one on the network who may be eavesdropping with something as simple as a network sniffer like snoop. Information obtained via these methods could be used to gain unauthorized access and control of hosts and user accounts, including root.

The following examples illustrate the vulnerabilities that are associated with the use of "r-commands" and just how simple it is to gain user account information from the network. The information in this example was captured during an rlogin attempt from the remote host 192.168.101 to the target host *spaceball2* using the network sniffer snoop. To ensure that the user attempting the rlogin was forced to enter a password no /etc/hosts.equiv or .rhosts entries existed. The exact command and an explanation of the arguments used to capture this information are as follows in examples 1 and 2:

Example1: Capturing Network Traffic

```
root# snoop -d elxl0 -o snoop.txt
```

Table 1A: Explanation of *Snoop* command arguments

Argument	Explanation
-d	Specifies the network interface to snoop on. In our case it is the interface elxl0
-o	Saves the output of the snoop command to a text file. In our example the output is sent to a file called snoop.txt

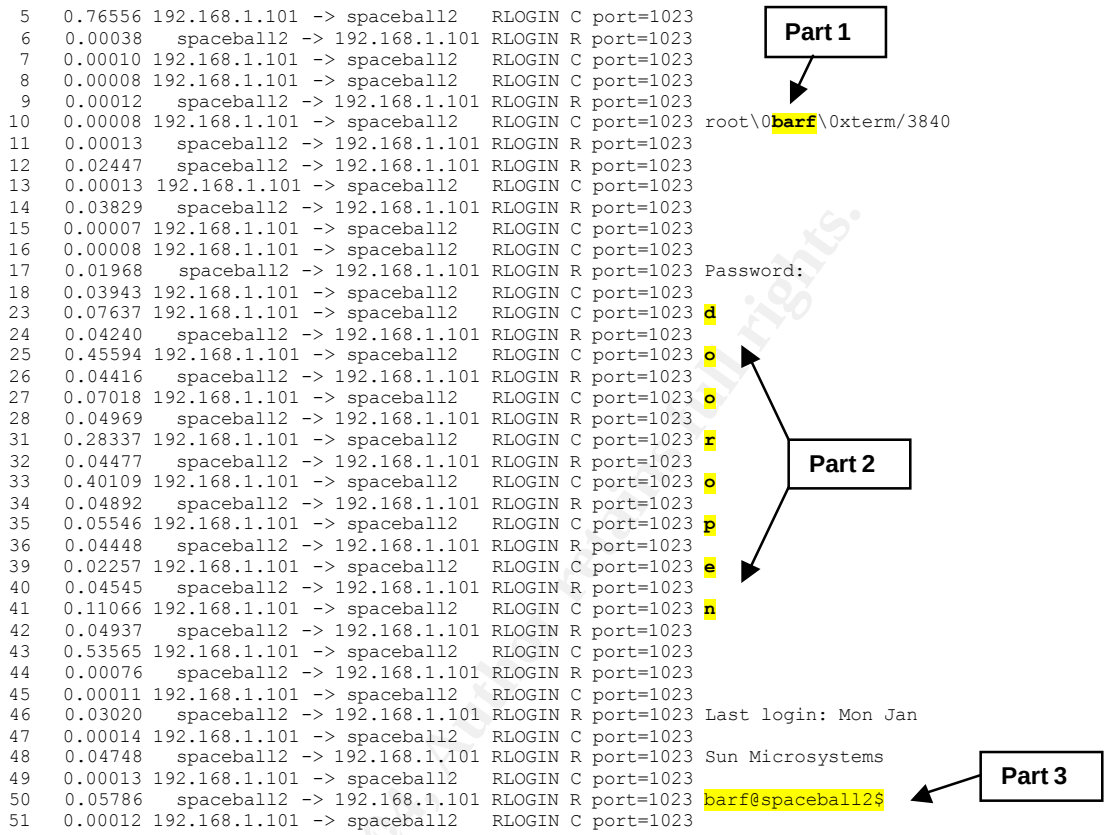
Example2: Read in Output from Text File

```
root#  snoop -i snoop.txt
```

Table 1B: More *Snoop* command arguments

Argument	Explanation
-i	Display the output from previously captured network traffic. In our example input is read from the text file snoop.txt

Example3: Excerpt of output from snoop session:




```
 5   0.76556 192.168.1.101 -> spaceball2   RLOGIN C port=1023
 6   0.00038   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
 7   0.00010 192.168.1.101 -> spaceball2   RLOGIN C port=1023
 8   0.00008 192.168.1.101 -> spaceball2   RLOGIN C port=1023
 9   0.00012   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
10   0.00008 192.168.1.101 -> spaceball2   RLOGIN C port=1023 root\0barf\0xterm/3840
11   0.00013   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
12   0.02447   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
13   0.00013 192.168.1.101 -> spaceball2   RLOGIN C port=1023
14   0.03829   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
15   0.00007 192.168.1.101 -> spaceball2   RLOGIN C port=1023
16   0.00008 192.168.1.101 -> spaceball2   RLOGIN C port=1023
17   0.01968   spaceball2 -> 192.168.1.101 RLOGIN R port=1023 Password:
18   0.03943 192.168.1.101 -> spaceball2   RLOGIN C port=1023
23   0.07637 192.168.1.101 -> spaceball2   RLOGIN C port=1023 d
24   0.04240   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
25   0.45594 192.168.1.101 -> spaceball2   RLOGIN C port=1023 o
26   0.04416   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
27   0.07018 192.168.1.101 -> spaceball2   RLOGIN C port=1023 o
28   0.04969   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
31   0.28337 192.168.1.101 -> spaceball2   RLOGIN C port=1023 r
32   0.04477   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
33   0.40109 192.168.1.101 -> spaceball2   RLOGIN C port=1023 o
34   0.04892   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
35   0.05546 192.168.1.101 -> spaceball2   RLOGIN C port=1023 p
36   0.04448   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
39   0.02257 192.168.1.101 -> spaceball2   RLOGIN C port=1023 e
40   0.04545   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
41   0.11066 192.168.1.101 -> spaceball2   RLOGIN C port=1023 n
42   0.04937   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
43   0.53565 192.168.1.101 -> spaceball2   RLOGIN C port=1023
44   0.00076   spaceball2 -> 192.168.1.101 RLOGIN R port=1023
45   0.00011 192.168.1.101 -> spaceball2   RLOGIN C port=1023
46   0.03020   spaceball2 -> 192.168.1.101 RLOGIN R port=1023 Last login: Mon Jan
47   0.00014 192.168.1.101 -> spaceball2   RLOGIN C port=1023
48   0.04748   spaceball2 -> 192.168.1.101 RLOGIN R port=1023 Sun Microsystems
49   0.00013 192.168.1.101 -> spaceball2   RLOGIN C port=1023
50   0.05786   spaceball2 -> 192.168.1.101 RLOGIN R port=1023 barf@spaceball2$
51   0.00012 192.168.1.101 -> spaceball2   RLOGIN C port=1023
```

The above example, example 3, has been broken into three parts to demonstrate the components of the rlogin attempt. In Part 1, you can see that an rlogin attempt to the user account *barf* was initiated from the source host at address 192.168.1.101 to the target host *spaceball2*. If the local user account *barf* exists on the hosts spaceball2 then a return prompt from the host *spaceball2* will be sent to 192.168.1.101 requesting a password. In Part 2 of this example, you can see the response from the user at hosts192.168.1.101 reply back with a password of *dooropen* for the user *barf*. If the password supplied by the user is correct then the rlogin will succeed for the user *barf* and the user will be presented with a prompt as seen in Part 3 of this example.

This brief example was a simple depiction of the type of information that when transmitted via clear text protocols can be captured from the network. You can imagine the consequences and severity of the problem if information for the root account were captured and then used to gain unauthorized access to a host. The ability to gain root access combined with the establishment of trusted hosts further emphasizes the risks associated with establishing trust relationships and provides greater justification for discontinuing their use.

Identified Risks

The use of .rhosts files and hosts.equiv files in conjunction with the "r-commands" creates a high level of risk because it could allow unauthorized access to user accounts including root, in addition to, allowing an attacker to gain unauthorized control of hosts within the enterprise. The risks that were identified are as follows:

- Inappropriate file permissions on root .rhosts, /etc/hosts.equiv and .rhosts in users home directories ($HOME/.rhosts) could allow unauthorized users to view or modify these files to gain unauthorized access.
- Incorrectly configured .rhosts and hosts.equiv files could grant excessive privileges (including root) to unauthorized users.
- A user ultimately controls .rhosts files in their home directory and could create a .rhosts file that grants access to whomever the user chooses without the system administrator's knowledge.
- Auditing and validating the trust relationships that exist and the extent of the privileges they grant is difficult because trust is transitive.
- The protocols used by the "r-commands" transmit data across networks in clear text making it easy to capture and use to gain unauthorized access.

The following examples give further illustration of the power and risks that trust configurations can introduce into an environment. The examples show how certain configurations combined with the use of wild cards, as previously discussed in the Problem Identification section, can easily create trust relationships that trust every host and user including root (Clayton, p1):

> ***Example 1:***
> The remote host spaceball has just a sole "+" entry within its local root level rhosts file (/.rhosts). Anyone user who has access to the root account on any workstation can read and write with rcp to the trusted host. An intruder could pull the password file, edit it and put it back without leaving much of trace of the activity.
>
> ***Example 2:***
> The remote host spaceball has "+ +" contained within its local root level rhosts file (/.rhosts). This entry allows any user from any machine to login into the host spaceball as the user root without a supplying a password.
>
> ***Example 3:***
> The remote host spaceball has within its /etc/hosts.equiv file has the entry ``lonestar +''. Here any user on the host lonestar can login as any other user on the host spaceball without having to supply a password.

The risks of trusted hosts have been well documented and it is understood as an industry best practices that these configurations should be avoided. To further illustrate the point on the dangers of these configurations, it was the notorious hacker Kevin Mitnick who exploited the weaknesses of trust relationships and the

"r-commands" in a successful attack against NEC cooperation. In that attack Kevin Mitnick used the finger command to identify a logged on user. A phone number was give as contact information from the output of the finger command. Kevin then contacted that user and convinced him to create a .rhosts file for him as part of some testing that he was conducting. Once the user created the rhosts file Kevin was able access that host and others within the NEC network (Goodwin, p1).

Existing Controls

In the original state of the environment, there were no centralized or standardized controls to govern the usage, monitoring of file contents or creation of root level .rhosts and /etc/hosts.equiv files. Additionally, there were no automated methods to ensure that the file permissions of those files were sufficient to prevent unauthorized modification. System administrators were not held accountable for the entries that were added to those files or for the monitoring of the privileges that occurred as a result of those established trust relationships.

Further complicating the situation was the division of responsibility for established trust relationships between two separate operational groups within the organization. The UNIX engineering team was tasked with the responsibility for configuration files that impacted a server at the global level (/etc/hosts.equiv) and the user files associated with the root account (/.rhosts). A second operational group within the organization, Data Security, was tasked with the responsibility for managing all users accounts and associated user files at that level ($HOME/.rhosts).

Self-Identified Gaps

A quick self-assessment of current process and procedures revealed that current controls were not effectively mitigating any of the risks posed by trust relationships. In most cases there was a lack of documentation and general lack of awareness by administrators and end users to what was trusted, by whom and for what reason. The absence of sufficient controls resulted in the following self-identified gaps:

- No accurate method for verifying and identifying all established trust relationships.
- No controls exist to ensure correct file permission on existing hosts.equiv and .rhosts files.
- No controls exist to validate the creation or deletion of hosts.equiv and .rhosts files.
- No controls exist to validate or report the contents of hosts.equiv and .rhosts files.

- No documentation exists to govern and standardize the use or prohibit the use of hosts.equiv and .rhosts files.
- No real time notification existed when one of these files is changed

Changing of The Environment: *The During*

Plan of Attack

As a member of UNIX engineering team I was the lead engineer tasked with responding to the audit findings and self identified gaps. As a starting point, a brain storming session was held with other engineers from the UNIX engineering and data security teams. This session identified six objectives to address the trust relationships and the identified gaps. The objectives, to be addressed in a phased approach, include identification, assessment, documentation, development, implementation and monitoring.

Identification of Trusted Hosts

The first step towards remediation and mitigation is identification. The scope of the problem had to be defined prior to us moving forward with any kind of corrective actions. One of the greatest challenges, working in an environment this large, was how to leverage the existing resources, technology and people, to obtain the required information. Ideally we wanted to avoid a situation where every systems administrator would have to log onto a server, look for specified conditions and send back a report. That type of process would be inefficient because it would require large amounts of time on the part of the administrator in addition to wasted effort by myself to correlate that information into a meaning full report. It was also reasonable to assume that this process would be repeated with regularity in the future, making it all the more important to develop an efficient mechanism from the beginning.

To accomplish this data collection task a few lines of code were added to an existing Korn shell script, that serves as one piece of an already existing in house configuration management compliancy tool. This tool runs on all Solaris servers on a daily basis. The additions to this script included new statements that used the UNIX find command with various arguments to identify .rhosts and /etc/hosts.equiv files on a per server basis. Once the changes were added to the Korn shell script, the new version of the script was propagated out to all serves in the enterprise via a pre-existing automated distribution method using NFS and SSH. After distributing the script throughout the environment, the script was run on all servers and the returned values were sent via an automated email to a centralized web server. Once all reports were run on all servers and delivered to the web server another in house script automatically correlated the individual server reports into one large report that was sorted by server name (More details on the in house scripts and web server will be addressed in the implementation

section of this paper). The following statements are examples of what was used to identify all .rhosts and hosts.equiv files (Logging In to a Remote System, docs.sun.com):

Example1: `#find / -name .rhosts -follow -exec ls -l{} >> $RESULTS`
Example2: `#find /etc -name hosts.equiv -follow -exec ls -l{} >> $RESULTS`

Table 2: Explanation of *Find* command arguments

Argument	Explanation
/ *or* /etc	Identifies the path to a directory where the search is to begin.
-name	Identifies the filename to be searched for. In this case it is either .rhosts or hosts.equiv.
-follow	Causes the underlying file of a symbolic link to be checked rather than the symbolic link itself.
-exec rm {}	Tells the *find* command to apply the *ls* command to all files identified using the matching filename.
>>	Redirects the output of the *find* command and appends it to a file.
$RESULTS	A variable used to define a text file that is used to capture the output of these statements

Note: These statements were part of a larger script which checked for more conditions than were related to this project. The overall contents of the file defined by the $RESULTS variable were sent to the centralized web server using mail.

The initial run of the security script revealed the full scope of the problem by identifying in excess of 500 instances of .rhosts and /etc/hosts.equiv files.

Assessment of Risks

The responsibility for the assessment of risks and vulnerabilities of previously identified .rhosts and hosts.equiv files will be divided between the two operational divisions in the organization. The assessment by the UNIX engineering services teams determined that the level of risks posed by currently established trust relationships at the systems and root level to be high. The data security team, responsible for user level configuration files and access, also determined the level of risk for user level .rhosts files ($HOME/.rhosts) to be high. The risk rating of high was determined from the following risks and vulnerabilities that could allow unauthorized access and control of hosts and user accounts, including root, within the corporate network.

- Inappropriate file permissions on root .rhosts and /etc/hosts.equiv files could allow unauthorized users to view or modify these files to gain unauthorized access. As detailed in the File Permissions and Ownership part of the Problem Identification section, correct permissions and ownership of .rhosts and /etc/hosts.equiv files are essential to limiting the amount of information that can be obtained by non privileged users. The ease in which excessive privilege can be granted system wide, to non-authorized users or systems, based simply on incorrect file permissions contributed to risk rating of high.

- Incorrectly configured .rhosts and /etc/hosts.equiv files could grant excessive privileges to unauthorized hosts and users including root. As discussed in detail in the File Configurations part of the Problem Identification section, proper file entries are critical to ensure that desired levels of privilege are being granted or denied. The ability to use the wild card character "+" to grant broad levels of systems wide access with a single entry in combination with incorrectly configured files contributed to the risk rating of high.

- Users ultimately have the control to create and modify .rhosts files in their home directory. With that control users are able to grant access to their local account to other users without the system administrator's knowledge or consent. Appropriate file permission, as discussed in File Permissions and Ownership part of the Problem Identification section, are essential for the "r-commands" to operate successfully. The ability of a user to grant privileges to other users without prior approval contributed the risk rating of high.

- The fact that trust operates in a transitive fashion means that privileges can be indirectly granted to systems and users that were not intentionally desired. As described in the Auditing and Validating Established Trust part of the Problem Identification section, the impact that directly trusted hosts can have on other non-trusted hosts contributed to the risk rating of high.

- The protocols used by the "r-commands" transmit data across networks, in clear text, which can be easily captured and used to gain unauthorized access and control of hosts and users including root. As previously demonstrated, in the Identified Vulnerabilities section, user login information is easily captured and could be used by an attacker. The information gathered from the network could be used to exploit other hosts and users including root. The ability to capture user login information directly of the network contributed the risk rating of high.

The data security team conceded that even though risks do exist, they were willing to accept those risks and permit .rhosts files at the user level in a limited capacity. The decision to continue to allow limited usage of user level .rhosts files was made because no immediate alternative solution for end users existed or would not be readily available within the time constrains of this project. Trust functionality needs to exist for certain application and user accounts that utilize the "r-commands" for functions such as batch processing jobs and application management tasks, similar to Trust Sample 2 in Figure2: Example Trust Relationships. Additional process and procedures will be put into place to assist in mitigating the risks that will continue to exist. A risk assessment report was drafted outlying the identified risks. Management with the understating that an alternative solution would be fully adopted in the future accepted the identified risks.

Development of Corrective Actions

Corrective actions were divided into two critical areas, remediation and mitigation with an emphasis on using Secure Shell as the desirable replacement for the "r-commands". The role that Secure Shell will serve along with its inherent benefits will be addressed in the next sections Using Secure Shell and Implementation of Corrective Actions.

Actions for Remediation

The goal of remediation was to remove the risks that resulted from trust relationships that were established, at the root and systems level, as a means of convenience for systems administrators, application owners and end users that were not required for valid business operations. As part of the remediation process the following corrective actions will be taken:

- Usage of root level .rhosts files (/.rhosts) and hosts.equiv (/etc/hosts.equiv) files will be prohibited. The files must exist and remain null (file size of zero) with the files being owned by root and permissions set to 400 (read only by root).

  Example: -r-------- 1 root other 0 Sep 27 10:25 .rhosts

  Example: -r-------- 1 root other 0 Sep 27 10:25 /etc/hosts.equiv

  The decision to maintain these files as opposed to there out right removal was based on trying to ensure that a user other than root could not create these files. While it was understood that the file permissions of the underlying file system should prevent the creation of these file by non-privileged users, the solution that generated the most support by management and by the system auditors was to maintain these file and enforce the above-mentioned restrictions in addition to monitoring for variances to the new standard

- Implementation of a documented standard outlying the restrictions that systems will need to comply too.

If any of the previously defined trust relationships cannot be removed and must exist in some capacity to fulfill a business need it will be addressed as part of the mitigation process. However, the level of privilege still cannot exist at the root level or at the systems level. The privilege must be reduced to the user level by replacing the /etc/hosts.equiv and root .rhosts entries with equivalent entries in users .rhosts files within their own home directories.

Actions for Mitigation

The goal of mitigation was to reduce the risk of trust relationships that continued to exist after the remediation phase had completed. The following courses of action will be taken as part of the mitigation process.

- Provide education and general awareness to the systems administrators, application owners and end users of the risks that these configurations create.

- Provide alternative solutions using Secure Shell (SSH) to achieve the same type of functionality that the original “r-commands provided”. Additionally, provide assistance in configuring SSH and help with explanations as to how and why SSH is a better solution. Further details on the benefits and features of Secure Shell will be discussed in the next section, Using Secure Shell.

- Data Security acknowledges that risk will continue to exist if user level .rhosts files are still to be permitted, even in a limited capacity. To mitigate those risks data security will enforce strict permissions on .rhosts files, monitor the contents of those files by looking for “+” characters and server names that are not fully qualified. Data security will also create a follow up process for contacting users when deviations are detected and provide assistance for correcting or finding alternative solutions. Data Security will also retain the authority to remove or enforce standards on entries as needed without user consent.

Using Secure Shell (SSH)

SSH is software approach to network security that is based upon the client/server architecture. SSH allows users to securely access remote hosts by providing transparent end-to-end encryption between the client and the server without interaction from the end user. Although the name would suggest that SSH is a shell, like Bourne or C shell, it actually is not a shell at all. SSH is a protocol that specifies how to conduct secure communication over a network. SSH creates a tunnel, using client and server programs, for running a shell on a remote computer and works in a similar manner to the UNIX command rsh. The distinct difference between rsh and SSH is that SSH provides the end-to-end encryption. In addition to the secure transmission of network traffic, SSH also provides authentication using passwords and public keys (Barrett, p2-4).

Secure Shell, for the purposes of this project, will be used as the desired tool for the replacement of the UNIX “r-commands”. SSH provides several direct replacements for the “r-commands” that will allow for secure remote logins, securely coping files between two hosts and running remote commands securely on a remote host. The SSH alternatives to rsh, rlogin and rcp are ssh, slogin, and scp respectively and use very similar syntax to the original “r-commands”. The use of similar syntax will further help to ease the learning curve for the new secure commands.

The authentication mechanisms supported by SSH are also of particular interest for this project. SSH supports authentication in two forms: password and public keys. If password authentication is used, it works in the same manner as the conventional “r-commands” except that user name and password information is encrypted prior to sending it across the network. The second method of

authentication, public keys, provides the very usefully benefit of not having to supply a password and would eliminate the need for the .rhosts and /etc/hosts.equiv files. A key is a digital identity that is composed of a unique string of binary data (Barrett, p26). The use of public keys allows for remote logins or execution of commands on a remote host with out having to supply a password while still being able to verify the identity of the user making the request. To verify identity SSH uses two parts, the private key and the public key to create what is referred to as a key pair (Barrett, p204). The private key is very important and should never be disclosed because an attacker who obtains your private key can gain access to systems using your identity. The private key is used by your SSH clients to prove your identity to an SSH server. The public key, on the other hand, does not need to be kept a secret and can be freely distributed into your user accounts for the systems you wish to access remotely. If an attacker were to obtain your public key, unlike the private key, they would be unable to access systems with your identity. Before using key based authentication the key pair must first be generated using the ssh-keygen program (Please see the note at the end of this section for references on SSH and key generation). Once the private key, your identity that resides on your client, and the public key, which resides on the server machine, have been generated you are then ready to do authentication via public keys.

> ***Note:*** Additional reference for SSH can be found using the book "SSH The Secure Shell" by O'Reilly. Also, additional information on the configuration of public and private keys can be found at: http://www.unixpeople.com/HOWTO/configuring.ssh.html

Development of Supporting Documentation

A new documented standard was developed to govern the usage of the files that establish trust-based relationships. Using templates and information found on SANS website (http://www.sans.org/resources/policies/) a new standard was written to govern the usage of trust-based relationships and the files that support them.

Highlights of Standard

- No use of .rhosts files should be found at the root level (/.rhosts) and no use of hosts.equiv files should be found in the /etc/ directory. These files should exist and be null with file permissions of 400 and owned by the user root.
- Monitoring procedures will be implemented to monitor for any usage of /.rhosts and /etc/hosts.equiv files on a daily basis. Results of the monitoring will be made available to system administrators daily with a summary report listing any findings being generated and reviewed monthly by management.
- If an occurrence is identified, the systems administrators will be notified by means of the configuration compliancy tool, responsible for providing the explanation for its use and for discontinuing its use or converting to one of the

two acceptable solutions (SSH or local user level rhosts files, $HOME/.rhosts).

Note: Although the original standard could not be used in this article, a sanitized copy of the original is located in *Appendix A*.

Implementation of Corrective Actions

To identify trusted hosts we settled on the idea of using existing automated home grown tools to identify and report on a per host basis the information on root level .rhosts and /etc/hosts.equiv files. Within the current environment every server deployed will have a mounted static copy of a filesystem that is common to all servers. The static filesystem is then updated on a regular basis through a combination of rsysnc and Secure Shell. Within this common file system, there exist a compilation of scripts and programs that constitutes a homegrown compliancy monitoring systems that reports to an internal web page. This internal web page is the primary interface between a systems administrator and the servers that they manage. The web page reports on log events, running processes, overall system health and deviations against imposed configuration standards. It is because of this functionality that we choose this existing tool to assist in our risk remediation and mitigation efforts.

Remediation Phase

The first phase of implementing corrective actions was the remediation phase in which all entries were removed from root level .rhosts files and /etc/hosts.equiv files. Systems administrators were given notice by email of all the established trust relationships on the servers they were responsible for managing. Also, appropriate change control requests were requested and approved by the enterprise change management board. The change control approval was required before systems administrators could begin making the required changes. A timetable of two weeks was allotted to have the changes completed for all 450 plus servers in the environment. During the two-week interval the following actions were conducted:

- A documented standard was implemented and communicated to administrators, business areas and end users. The information was presented in the form of email and by communicating the new changes during weekly team meetings. The documented standard was also published on an internal web site along with other documented standards, procedures and polices.

- Systems administrators made the necessary configuration changes, removing entries in root .rhosts and /etc/hosts.equiv files, during approved maintenance windows. Assistance was provided to administrators who required assistance with the alternative solution Secure Shell and configuring user level rhosts files. Secure Shell as a solution offers similar functionally to the UNIX "r-commands" and provides transparent end-to-end encryption of network traffic between hosts. In addition, the convenience that "r-commands"

provided by not having to supply passwords can be achieved in a more secure fashion with the use of SSH and host-based keys. For more details on how SSH works as a solution and the benefits it provides, consult the Using Secure Shell section of this paper.

At the end of the two-week time period a report was run, equivalent to the one used in the identification phase, and the results reviewed. Analysis of the report showed less than five servers out of the approximated 450, to be out of compliance. Those remaining servers were addressed on an individual basis to achieve compliance.

Mitigation Phase

The second phase of implementing corrective actions was the mitigation phase in which all-remaining .rhosts files, only user level .rhosts should be left, were addressed. The efforts of data security were mostly directed in the area of user .rhosts files. Not being a member of the data security, I was not as involved in the details or the actual "hands-on" portion of this phase. However, the details of the actions that were taken are outlined here:

- Provide education and general awareness to the systems administrators, application owners and end users of the risks that these configurations create in a manner similar to what was done in the remediation phase.
- Provided alternative solutions using SSH to achieve the same type of functionality that the original "r-commands provided". In addition, provide assistance in configuring SSH and help with explanations as to how and why SSH is a better solution.
- New user accounts that get created will have a .rhosts file created with the permissions set to read-only (400) and null (file size of zero).
- Data security will enforce strict permissions on .rhosts files by monitoring for any .rhosts file that is readable by "other" (read-only for the owner and group).
- The contents of .rhosts files will also be monitored for occurrences of "+" characters and server names that are not fully qualified.
- Data security also created a follow up process for contacting users when deviations are detected and to provide assistance for correcting or finding alternative solutions.
- Data Security retains the authority to remove or enforce standards on entries as needed without user consent.

Compliancy Monitoring

Upon completion of the remediation and mitigation phases the environment now possessed less risks than prior to the corrective actions. It is important to realize that our efforts did not end after all the corrective actions had been completed. In actuality a new living breathing process had been born that will continually need to be addressed. Monitoring for compliancy of the newly imposed standard is a very crucial step towards achieving long-term success. Management will use the reports that are generated as a tool to gauge effectiveness of administrators and to measure corporate wide compliancy to the newly defined standard. In addition, effective compliancy monitoring is required to demonstrate to any follow up audit teams the state of the environment at any given point in time.

The tool chosen for compliancy monitoring is an in house homegrown application that is the central management tool used by systems administrators and management to administer serves and to report on server statistics and compliancy to standard configurations. The homegrown tool centers on an apache web server and a compilation of shell scripts (Korn, Bourne and Perl) that gather information from all the servers in the environment and provides that information in a graphical representation to the web server. The details and intricacies of this tool are too vast to go into detail for the purposes of this paper.

Compliancy for trust relationships at the root and systems level will be checked daily on a per server basis as required to fulfill imposed corporate reporting requirements. Administrators will be alerted, via an online notification, of any deviations from the imposed standard and will be tasked with the appropriate corrective actions as outlined in the standard. Failure to correct any deviations will be reflected in monthly summary reports, required by management, that outline compliancy to all imposed standards. The reports sent to management will be used as a tool to measure success in achieving compliance to imposed standards and the effectiveness of administrators in resolving any deviations.

As part of the daily monitoring process a new routine was added to an already existing shell script to ensure that permissions on root .rhosts files and /etc/hosts.equiv files were correct. The new routine ensures the user *root* with a group of *other* always owns these files. The shell script also ensures that the files permissions are set to read-only by the root user (permission of 400). In addition, a software check was put into place to make sure that the file size is always zero (null). If any one of the files has a size other than zero it is reported and a notification is sent to the applicable systems administrator noting that follow up action is required. An excerpt of the compliancy shell script that deals with trust files and an explanation of the statements within the routine are shown in the following example:

Example1: Excerpt from Compliancy Shell Script

```
#-------------------------------------------
#        Security Checks for Trust Files
#-------------------------------------------
for FILE in ~/.rhosts /etc/hosts.equiv
      do

            if [ ! -f $FILE ]
            then
Part 1            /usr/bin/touch $FILE
            fi

            /usr/bin/chown root $FILE >/dev/null 2>&1
Part 2      /usr/bin/chgrp other $FILE >/dev/null 2>&1
            /usr/bin/chmod 400 $FILE >/dev/null 2>&1

            if [ -s $FILE ]
            then
Part 3            echo "\n$FILE should be empty:" >> $RESULTS
                  /usr/bin/ls -lL $FILE  >> $RESULTS
            fi

      done
```

The *for* loop routine in this example is just one part of a much larger script that checks for more conditions then related to this project. The structure of the *for* statement will allow for each of the statements contained within it to be executed for each of the specified files. Part 1 of this example checks to ensure that the files .rhosts and hosts.equiv exist. If the file does not exist it is created. In the second part of this routine, Part 2, The UNIX commands, chown, chgrp and chmod will be used to ensure proper file permissions and file ownership with any output and standard error being directed to /dev/null. Part 3 of the example checks to make sure the files .rhosts and hosts.equiv have a file size of zero (null). If the file is not empty a notification is sent to the systems administrator indicating corrective action is required.

Note: These statements were part of a larger script which checked for more conditions than were related to this project. The overall contents of the file defined by the $RESULTS variable were sent to the centralized web server using mail.

Compliancy for trust relationships at the user level will be checked monthly on a per server basis by a script written by the data security team. A member of the data security team will notify users who are found to be in violation of the imposed standard. Users will be responsible for correcting the problem or requesting assistance, if needed, from data security. Monthly summary reports, as required by data security management, will outline compliancy to all imposed standards and will be used to track compliancy over time.

New State of The Environment: *The After*

Success or Failure

The project was a success as we addressed the deficiencies identified by the internal audit team and the gaps from the self-assessment in the time that was allotted for this project. The original state of the environment had approximately 500 wrongfully configured or misused .rhosts and .hosts.equiv files. As a result of our efforts and addressing the project in a phased approach we were able to identify and assess the risks, determine the full scope of the problem, develop and implement solutions that allowed us to successfully eliminate root level .rhosts and /etc/hosts.equiv entries to the extent that only about a half dozen remained that would have to be addressed on an individual basis. We were equally successful in leveraging existing tools and processes to implement effective compliancy monitoring procedures to ensure long-term compliancy to the new standards. The removal of unnecessary entries, about 90 percent of total number, dramatically reduced the level of risk to the environment prior to our efforts. Pushing the level of privilege of trusted relationships to the user level and or using alternative solution like Secure Shell further reduced the level of risk. The use of SSH will ultimately allow for the complete elimination of user level .rhosts files and the eventual disablement of the “r-commands”.

Remaining Gaps and Risks

Upon completion of this project some gaps and risks continued to exist even after corrective actions were completed. The remaining gaps and risks were identified as:

- Users are still permitted to use and control .rhosts files in their home directories ($HOME/.rhosts)
- No method for receiving real-time alerts when a change is made to .rhosts or hosts.equiv files. Some form of event based notification solution will have to be evaluated in the future.
- Secure Shell (SSH) is not fully adopted throughout the environment. Without having SSH fully adopted throughout the enterprise it was not feasible to fully prohibit users from using “r-commands” and their need for .rhosts files. The complexity of the environment will make it difficult to ensure SSH is available for all platforms and versions.
- The “r-commands” are still available for use even if .rhosts and hosts.equiv files are not used. The risk that this present is that although users are forced to supply a password the information is still transmitted via clear text protocols. SSH is a suitable replacement for this situation.

Assessment of Remaining Gaps and Risks

Although risks and some gaps still existed after completion of the corrective actions a follow up by the internal audit team concluded that the level of risks to the environment had been reduce to a sufficient level and that satisfactory risk remediation and mitigation had been done to improve the overall security posture of the environment.

The only way to fully mitigate the risks posed by trusted relationships and the "r-commands" would be to not permit their use. However, that is easier said then done and in most cases improving the security of an environment becomes a balancing act of required functionality and the time and cost it takes to implement improved security measures. In our particular instance the ideal solution would be the full adoption of Secure Shell. Secure Shell would take the place of the "r-commands" and would allow them to be disabled through the plugable authentication module (PAM) or completely removed from the operating system.

Appendix A: *Standard for Trust Relationships*

Company Confidential

Computer & Network Operations
Definition of Standard
Standard: CNO-SD001
Trust Relationships

Policy Ref#: SEC-PL001

Last Revision April 24, 2003

Purpose

The purpose of this standard is to establish guidelines and procedures for the base configuration of internal server equipment that is owned and or operated by Acme Corporation. Effective implementation of this standard will minimize unauthorized access to Acme Corporate proprietary information and technology and ensure compliance with established Acme Corporate policy.

Established Acme Corporate policy governing this standard can be obtained from the internal Acme website. Located at http://www.acme.com/iss

Scope

This standard applies to server equipment owned and or operated by Acme Corporation, In addition to any servers registered under any Acme Corporation owned internal network domain.

Definition of Trust Relationship

A trusted relationship is created when the normal standard password-based user authentication mechanism is bypassed. The remote authentication procedure determines whether a user from a remote host should be allowed to access the local system with the identity of a local user. Authentication is then granted or denied based upon policies explicitly set to allow users remote privileges with out verifying their credentials.

Standard

Restrictions for /.rhosts files at the root level

Usage of any of the remote services (rlogin, rsh, rpc, etc) at the root level will be strictly forbidden. No usage of /.rhosts files should found at the root level (/.rhosts).

Monitoring procedures will be implemented to monitor for any usage of /.rhosts files. A report listing any findings will be generated and reviewed monthly. If an occurrence is identified the following actions will be taken:

1. The applicable system administrator will be notified of the occurrence by management
2. The systems administrator will be responsible for providing justification for its usage.
3. The systems administrator will be responsible for providing a time frame for discontinuing usage or for the conversion to a better practice.

Functionality for authentication of any of the existing remote services (rlogin, rsh, rpc, etc) should be removed. There are two acceptable courses of action that need to be taken:

1. If the functionality is not required by the business area or the customer, then perform the following (Preferred Course of Action):

Remove all entries within the /. rhosts file.
Ensure that the ownership of /.rhosts file is root:other
Ensure that the permissions of the /.rhosts file is 400 (read by root only)

Example: -r-------- 1 root other 0 Sep 27 10:25 .rhosts

2. If the functionality provided by remote services is required by the business area or customer, then one of the following courses of action must be taken:

   An alternative solution must be implemented to replace the use of the remote services. SSH may be a valid solution for meeting both the security requirements while at the same time providing necessary functionality to the customer.

   Reduce the level of privilege of the remote services by replacing the root level /. rhosts file with an equal entry in the applicable users home directory. This solution should be used only if the remote service is a business or customer requirement.

Restrictions for hosts.equiv files
Usage of /etc/hosts.equiv files will be strictly forbidden. No usage of /etc/hosts.equiv files should found at the system level.

Monitoring procedures will be implemented to monitor for any usage of /etc/hosts.equiv files. A report listing any findings will be generated and reviewed monthly. If an occurrence is identified the following actions will be taken:

1. The applicable system administrator will be notified of the occurrence by management.
2. The systems administrator will be responsible for providing justification for its usage.
3. The systems administrator will be responsible for providing a time frame for discontinuing usage or for the conversion to a better practice.

Functionality for authenticating any of the existing remote services (rlogin, rsh, rpc, etc) should be removed. There are two acceptable courses of action that need to be taken:

1. If the functionality is not required by the business area or the customer, then perform the following (Preferred Course of Action):

   Remove all entries within the /etc/hosts.equiv file.
   Ensure that the ownership of /etc/hosts.equiv file is root:other
   Ensure that the permissions of the /etc/hosts.equiv file is 400 (read by root only)

   Example: -r-------- 1 root other 0 Sep 27 10:25 /etc/hosts.equiv

If the functionality provided by remote services is required by the business area or customer, then one of the following courses of action must be taken:

1. An alternative solution must be implemented to replace the use of the remote services. SSH may be a valid solution for meeting both the security requirements while at the same time providing necessary functionality to the customers.
2. Reduce the level of privilege of the remote services by replacing the /etc/hosts.equiv with a .rhosts entry in the applicable users home directory. This solution should be used only if the remote service is a business or customer requirement

Monitoring

Monitoring for compliancy as well as violations of this standard will be accomplished by means of our standard monitoring procedures and compliancy process.

Enforcement

Any employee found to have violated or not adhered to the established standard and guidelines might be subject to disciplinary action as mandated by established Acme policy.

Definitions

TERM	DEFINITION/EXPLANATION
SSH	Secure Shell

Revision History

REVISED BY	REVISION DATE	EXPLANATION OF CHANGE
John Doe	April 23, 2003	Finial Revision

Works Cited

Andert, Donna, Wakefield, Robin and Weise, Joel. "Trust Modeling for Security Architecture Development." Sun BluePrints OnLine December 2002: 2. URL: <http://www.sun.com/solutions/blueprints/1202/817-0775.pdf>.

"Authentication for Remote Logins." Solaris 9 System Administrator Collection: System Administration Guide: Resource Management and Network Services. Sun Microsystems, Inc: http://docs.sun.com 2002. URL: <http://docs.sun.com/db/doc/806-4076/6jd6amrd7?a=view>.

Barclay, Andy. "Configuring SSH". http://www.UnixPeople 2002. URL: <http://www.unixpeople.com/HOWTO/configuring.ssh.html>.

Barrett J, Daniel and Silverman E, Richard. SSH The Secure Shell. Sebastopol: O'Reilly & Associates, Inc., 2001.

Clayton A, C. "Starlink System Note 37.1." Starlink Security. Starlink Project: http://www.starlink.rl.ac.uk May 1996. 1-2. URL: <http://www.starlink.rl.ac.uk/star/docs/ssn37.htx/node24.html>.

Cole, Eric, Millican M, John and Newfield, Mathew. GSEC Security Essentials Toolkit. Indianapolis: Que Publishing, 2002. 41-42.

Goodwin, Bill. "Hacker Mitnick gives it to you straight". http://www.computerweekly.com January 23 2003. 1. URL: <http://www.computerweekly.com/Article118786.htm>.

Gregory H, Peter. Solaris Security. Upper Saddle River: Prentice Hall PTR, 2000. 169-170.

"Hosts.equiv(4)." Solaris 9 Reference Manual Collection: man pages section 4: File Formats: hosts.equiv(4) - trusted remote hosts and users. Sun Microsystems, Inc: http://docs.sun.com June 23,1997. URL: <http://docs.sun.com/db/doc/816-0219/6m6njqb8i?q=hosts.equiv&a=view

"Logging In to a Remote System." Solaris 2.6 System Administrator Collection Vol 1 : System Administration Guide: Part XI Working With Remote Systems. Sun Microsystems, Inc: http://docs.sun.com 1997. URL: <http://docs.sun.com/db/doc/802-5750/6i9g464od?a=view>.

Last Updated: January 22nd, 2019

Upcoming Training

SANS Las Vegas 2019	Las Vegas, NV	Jan 28, 2019 - Feb 02, 2019	Live Event
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
Security East 2019 - SEC401: Security Essentials Bootcamp Style	New Orleans, LA	Feb 04, 2019 - Feb 09, 2019	vLive
SANS Northern VA Spring- Tysons 2019	Tysons, VA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Anaheim 2019	Anaheim, CA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Dallas 2019	Dallas, TX	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Secure Japan 2019	Tokyo, Japan	Feb 18, 2019 - Mar 02, 2019	Live Event
SANS Scottsdale 2019	Scottsdale, AZ	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS New York Metro Winter 2019	Jersey City, NJ	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Reno Tahoe 2019	Reno, NV	Feb 25, 2019 - Mar 02, 2019	Live Event
Open-Source Intelligence Summit & Training 2019	Alexandria, VA	Feb 25, 2019 - Mar 03, 2019	Live Event
Mentor Session @Work - SEC401	Raleigh, NC	Feb 27, 2019 - Mar 06, 2019	Mentor
SANS Baltimore Spring 2019	Baltimore, MD	Mar 02, 2019 - Mar 09, 2019	Live Event
Baltimore Spring 2019 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Mar 04, 2019 - Mar 09, 2019	vLive
Community SANS Indianapolis SEC401	Indianapolis, IN	Mar 04, 2019 - Mar 09, 2019	Community SANS
SANS Secure India 2019	Bangalore, India	Mar 04, 2019 - Mar 09, 2019	Live Event
SANS Secure Singapore 2019	Singapore, Singapore	Mar 11, 2019 - Mar 23, 2019	Live Event
SANS London March 2019	London, United Kingdom	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS San Francisco Spring 2019	San Francisco, CA	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS St. Louis 2019	St. Louis, MO	Mar 11, 2019 - Mar 16, 2019	Live Event
Mentor Session - SEC401	Fredericksburg, VA	Mar 12, 2019 - May 14, 2019	Mentor
SANS Secure Canberra 2019	Canberra, Australia	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS Norfolk 2019	Norfolk, VA	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS Munich March 2019	Munich, Germany	Mar 18, 2019 - Mar 23, 2019	Live Event
SANS vLive - SEC401: Security Essentials Bootcamp Style	SEC401 - 201903,	Mar 19, 2019 - Apr 25, 2019	vLive
Community SANS Raleigh SEC401	Raleigh, NC	Apr 01, 2019 - Apr 06, 2019	Community SANS
SANS 2019 - SEC401: Security Essentials Bootcamp Style	Orlando, FL	Apr 01, 2019 - Apr 06, 2019	vLive
SANS 2019	Orlando, FL	Apr 01, 2019 - Apr 08, 2019	Live Event
SANS London April 2019	London, United Kingdom	Apr 08, 2019 - Apr 13, 2019	Live Event
Blue Team Summit & Training 2019	Louisville, KY	Apr 11, 2019 - Apr 18, 2019	Live Event
SANS Riyadh April 2019	Riyadh, Kingdom Of Saudi Arabia	Apr 13, 2019 - Apr 18, 2019	Live Event