Overview

This article describes the architecture, details and instructions for setting up restricted shell access.

For many CSE related virtual labs, it is a fundamental requirement to provide a way for the user to interact with a computer. This interaction could be the execution of system command to observe the output, the execution of a program to see the result or the execution of a program by an evaluation mechanism. It is desirable to make this interaction imitate a real system interaction as closely as possible. Hence providing a terminal emulator on the user's browser while taking care of security considerations would be a good solution.

This document is intended for lab developers looking for a way to use the remote shell service and for system administrators maintaining the infrastructure for restricted shell access.

Architecture

The following is the currently agreed upon architecture for providing restricted shell access.

Working:

1. User visits the virtual lab webpage
2. The webpage loads some scripts
3. The script makes a connection to backend 'Gateone' gateway server using WebSockets
4. The gateway server then connects to a backend SSH server via the SSH protocol
5. The gateway server also emulates a terminal to the SSH server
6. The contents of the emulated terminal are sent to the user's web page and get displayed
7. All the user's input is forwarded via the gateway the SSH server
8. For security, there are several limitations put up on the user's account in the restricted shell SSH server
9. The user account information comes from an OpenLDAP server setup separately
10. Accounts on the OpenLDAP server get created when the user registers for virtual labs
11. Home directories for the user's are on a NFS mount which is served by a central NFS server with quotas setup

Restrictions

Restricting the resource usage of a user is critical because a user could inadvertently hog all the server resources. A plan to create a separate container or virtual machine for each user seems to require a lot of server resources making it untenable when the setup scales. Hence, the plan is to create a container that is unrestricted and host many users in a single container.

Then, resources are restricted on a per-user basis. This is done using the traditional security limits on a GNU/Linux system. These limits are set in the **/etc/security/limits.conf**. See man page **limits.conf(5)** for more information. The virtual labs template to create the restricted shell containers creates the following limits by default:

```
@vlusers - core 0           # Core dump file size
@vlusers - data 10240       # Size of program data area
```

```
@vlusers - fsize 102400     # Size of a file
@vlusers - memlock 10240    # Size of locked memory
@vlusers - nofile 128       # No. of open files/sockets
@vlusers - cpu 1            # CPU time in minutes
@vlusers - nproc 20         # No. of simultaneous processes
@vlusers - as 102400        # Size of the address space
@vlusers - maxlogins 2      # No. of allowed logins
@vlusers - priority 0       # Nice value of the processes
@vlusers - locks 64         # No. of file locks
@vlusers - sigpending 64    # Size of the pending signal queue
@vlusers - msgqueue 102400  # Size of IPC message queue
@vlusers - stack 10240      # Size of the program call stack
```

Gateone Server

Gateone server provides a terminal emulator and ssh connection from a web browser. It does so by connecting the web client to a backend server using HTML5 WebSockets and then proxying a real SSH connection to another server. Gateone is licensed under GNU Affero General Public License version 3 is and available for download. A .deb package is available and can be installed directly on an Debian/Ubuntu server.

Containers

Shell access to a server increases the risk of server security breach. Evaluating programs written by the users is also a dangerous thing for the server. Isolating the shell servers is then the logical things to improve security. This can either be done using a virtual machine or using Linux Containers (LXC). Gateone shall run as separate container and each of the shell servers will run as a container either on same hardware or on different hardware.

Configuration and setup of containers is non-trivial. So a tool to create these containers has been developed. It is available as an LXC template from its Subversion repository:

```
svn+ssh://svn.virtual-labs.ac.in/labs/cse09/lxc
```

Evaluation Server

Evaluation of programs written by the user may be provided as a service depending on labs. Evaluation server has some test cases that are not public knowledge. This is somewhat like a examination paper. For this, the evaluation service is provided as a web service where user may upload the program and get results. The evaluation server in turn must use restricted shell server to evaluate programs. Otherwise it runs the same security risks that a SSH server for user experimentation runs.

Hence evaluation server must use the same restrictions as the shell servers. The programs are then run using the same user account as the user who has uploaded her program. Restrictions such as CPU time limit, maximum no. of file descriptor limit etc. apply. These servers have to separate from the restricted shell server in order for users not to be able to contaminate the execution process or results.

Setup Instructions

Central User Accounts (LDAP)

On the server

- Install OpenLDAP server

```
# apt-get install slapd
```

- Answer the following to the debconf questions

```
Omit OpenLDAP server configuration? No
DNS domain name: virtual-labs.ac.in
Organization name? Virtual Labs
Administrator password: password
Confirm password: password
Database backend to use: HDB
Do you want the database to be removed when slapd is purged? No
Allow LDAPv2 protocol? No
```

- Verify that the OpenLDAP server is setup properly

```
# ldapsearch -Y EXTERNAL -H ldapi:// -b 'dc=virtual-labs,dc=ac,dc=in'
```

You should see entries some what like this:

```
# extended LDIF
#
# LDAPv3
# base <dc=virtual-labs,dc=ac,dc=in> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# virtual-labs.ac.in
dn: dc=virtual-labs,dc=ac,dc=in
objectClass: top
objectClass: dcObject
objectClass: organization
o: Virtual Labs
dc: virtual-labs

# admin, virtual-labs.ac.in
dn: cn=admin,dc=virtual-labs,dc=ac,dc=in
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator
```

- Create organizational units for people and groups

```
# ldapadd -x -D 'cn=admin,dc=virtual-labs,dc=ac,dc=in' -W -f units.ldif
```

units.ldif should look like this:

```
dn: ou=People,dc=virtual-labs,dc=ac,dc=in
ou: People
objectClass: organizationalUnit
```

```
dn: ou=Group,dc=virtual-labs,dc=ac,dc=in
ou: Group
objectClass: organizationalUnit
```

- Create a group 'vlusers' for Virtual Labs end users

```
# ldapadd -x -D 'cn=admin,dc=virtual-labs,dc=ac,dc=in' -W -f group.ldif
```

group.ldif should look like this:

```
dn: cn=vlusers,ou=Group,dc=virtual-labs,dc=ac,dc=in
cn: vlusers
gidNumber: 20000
objectClass: top
objectClass: posixGroup
```

- Create a 'testuser' user in 'vlusers' group

```
# ldapadd -x -D 'cn=admin,dc=virtual-labs,dc=ac,dc=in' -W -f testuser1.ldif
```

testuser1.ldif should look like this:

```
dn: uid=testuser1,ou=People,dc=virtual-labs,dc=ac,dc=in
uid: testuser1
uidNumber: 20000
gidNumber: 20000
cn: Test User 1
sn: User
objectClass: top
objectClass: person
objectClass: posixAccount
objectClass: shadowAccount
loginShell: /bin/bash
homeDirectory: /home/testuser1
```

- Check that everything got created properly using the search command above.

On the shell machines

- Install the libpam-ldapd package

```
# apt-get install libpam-ldapd
```

Answer as follows to the questions

```
IP address / hostname of the LDAP server: ldap.virtual-labs.ac.in
The search base: dc=virutal-labs,dc=ac,dc=in
Version of the LDAP connecting to: Version 3
Configuring LIBNSS-LDAP: OK
Make root the DB admin: Yes
DB requires logging in: No
Root account of LDAP: cn=admin,dc=virtual-labs,dc=ac,dc=in
Root password: password
```

- Modify /etc/nsswitch.conf to contain something like this:

```
passwd:     files ldap
group:      files ldap
```

```
shadow:     files ldap
```

- Verify that the LDAP server is being reached and everything is working:

```
getent passwd
```

See some entries getting added at the end like this:

```
testuser1:x:20000:20000:Test User 1:/home/testuser1:/bin/bash
```

- Enable creating home directories when user logs in. Edit /etc/pam.d/common-session and add the following line

```
session     required      pam_mkhomedir.so skel=/etc/skel umask=0022
```

NFS Home directories

On the server

- Have a separate partition for hosting NFS home directories and setting up Quotas such as /var
- Create a directory to keep NFS home directories. This has to be on a separate filesystem ideally where quotas can be enabled.

```
# mkdir -p /var/export/nfs4/home
```

- Install NFS kernel server

```
# apt-get install nfs-kernel-server
```

- Edit /etc/exports and add the following lines (set IP subnet properly)

```
/var/export/nfs4       10.2.48.0/24(rw,sync,no_subtree_check)
/var/export/nfs4/home  10.2.48.0/24(rw,sync,no_subtree_check)
```

- Refresh the exports list

```
# exports -rav
```

- Install Quota tools

```
# apt-get install quota quotatools
```

- Insert the kernel module for quotas version 2

```
# modprobe quota_v2
```

- Create quota files

```
# touch /var/aquota.user /var/aquota.group
```

- Mark the filesystem for quotas by editing and adding usrquota and grpquota options in /etc/fstab

```
/dev/mapper/vg01-var /var            ext4    defaults,usrquota,grpquota        0       2
```

- Scan the filesystem to update quota information

```
# quotacheck -m -F vfsv1 -u -g /var/
```

- Edit the quotas for the group in which virtual lab user are present

```
# edquota -g vlusers -F vfsv1
```

- Turn on quotas

```
# /etc/init.d/quota start
```

On a non-container client

- Edit /etc/fstab and add the following line (with proper server address)

```
10.2.48.10:/var/export/nfs4/home /home nfs4 defaults 0 1
```

- Mount the filesystem now. This is happen automatically if the system reboots.

```
# mount -a
```

On a container client

- Edit /etc/lxc/eval1.conf and add the following line (where 'eval1' is name of the container)

```
lxc.mount.entry = /var/export/nfs4/home /var/lib/lxc/eval1/rootfs/home none rw,bind 0 0
```

- Stop and start the container

```
# lxc-stop -n eval1
# service lxc start
```