

Project 3 System Security

Sharath Bangalore Ramesh Kumar (sbangal2)

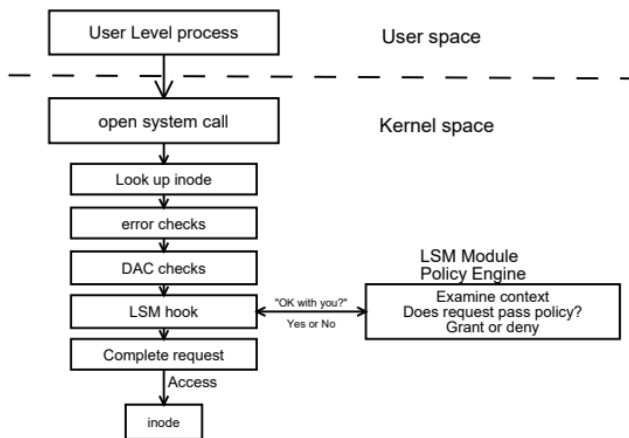
QUESTION 1:

1. For your write-up of this question, provide a short documentation for how your LSM functions. Explain when the hooks are called and why it meets the criteria specified for PinDOWN. Additionally, if you have made enhancements to the design, explain them here.

Solution:

Linux Security Model (LSM) provides a general-purpose framework for security policy modules. The LSM provides a framework that allows the linux kernel to support to various computer security models without emphasizing on a single security implementation. LSM provides linux kernel with a general-purpose framework for access control policies. LSM enables loading enhanced security policies as a kernel module. This allows many different access control models to be implemented as loadable kernel modules, enabling multiple threads of security policy engine development to proceed independently of the main linux kernel. For implementing various access control policies LSM provides number of hooks. The LSM framework must be truly generic, conceptually simple and capability to support existing security models.

LSM Hook Architecture:



Flowchart Reference:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.5163&rep=rep1&type=pdf>

From the above given flowchart, the User Level Process in the user space will execute the system calls. The system call will traverse the existing linux kernel logic for finding and allocating the resources, then perform error checking and passing the access control. The LSM hook will be executed at the end of the hierarchy, just before the kernel attempts to access the internal objects.

The Pindown uses linux existing **setfattr** and **getfattr** command to set and view security policies.

As the command given in the project 3 description pdf:

```
setfattr -n "security.pindown" -v <editor used to open the file (vim/gedit/vi etc.)>  
<File for which the security policy needs to be applied>
```

For example, when a particular file (for ex: /home/sbangal2/text.txt) should have access policy by a specific application (for ex: /usr/bin/vim), then the above command:

```
setfattr -n "security.pindown" -v "/usr/bin/vim" "/home/sbangal2/text.txt"
```

In the above-mentioned example, the extended attribute is **security.pindown** and the extended attribute value is **/usr/bin/vim**. The filename or the inode in this case is **/home/sbangal2/text.txt**. The file text.txt can only be accessed by the **vim** application.

The Pindown functionality is implemented with the help of following functional hook:

1. task_alloc_security()
2. task_free_security()
3. brpm_set_security()
4. inode_permission()

task_alloc_security()

For example, using setfattr the extended attribute for /home/sbangal2/text.txt is set to /usr/bin/gedit. When the gedit application is run, the operating system will fork the process and it hits task_alloc_security() function and sets the security field with path name /usr/bin/gedit and its path length. Task_alloc_security() is called on fork.

brpm_set_security()

While the gedit is loading its program, the gedit application will call exec(). On calling the exec(), brpm_set_security() is invoked. The brpm_set_security() captures the current running binary in the brpm_set_security hook. This is called when the binary program is loaded.

Inode_permission()

The primary access control decision is made by the inode_permission() hook. The inode_permission() hook is called during access of the file. This hook will check the permission before accessing the inode (for example a file). The structure of the running process is obtained by using the current variable. If the inode doesn't have the xattr parameter, then we have to allow the file to be accessed. If the file has xattr for "security.pindown", the file will be accessed only when the value of the string xattr matches with pathname store in the brpm_pathname of gedit application. If the match is not found, then the access to the file is denied.

task_free_security()

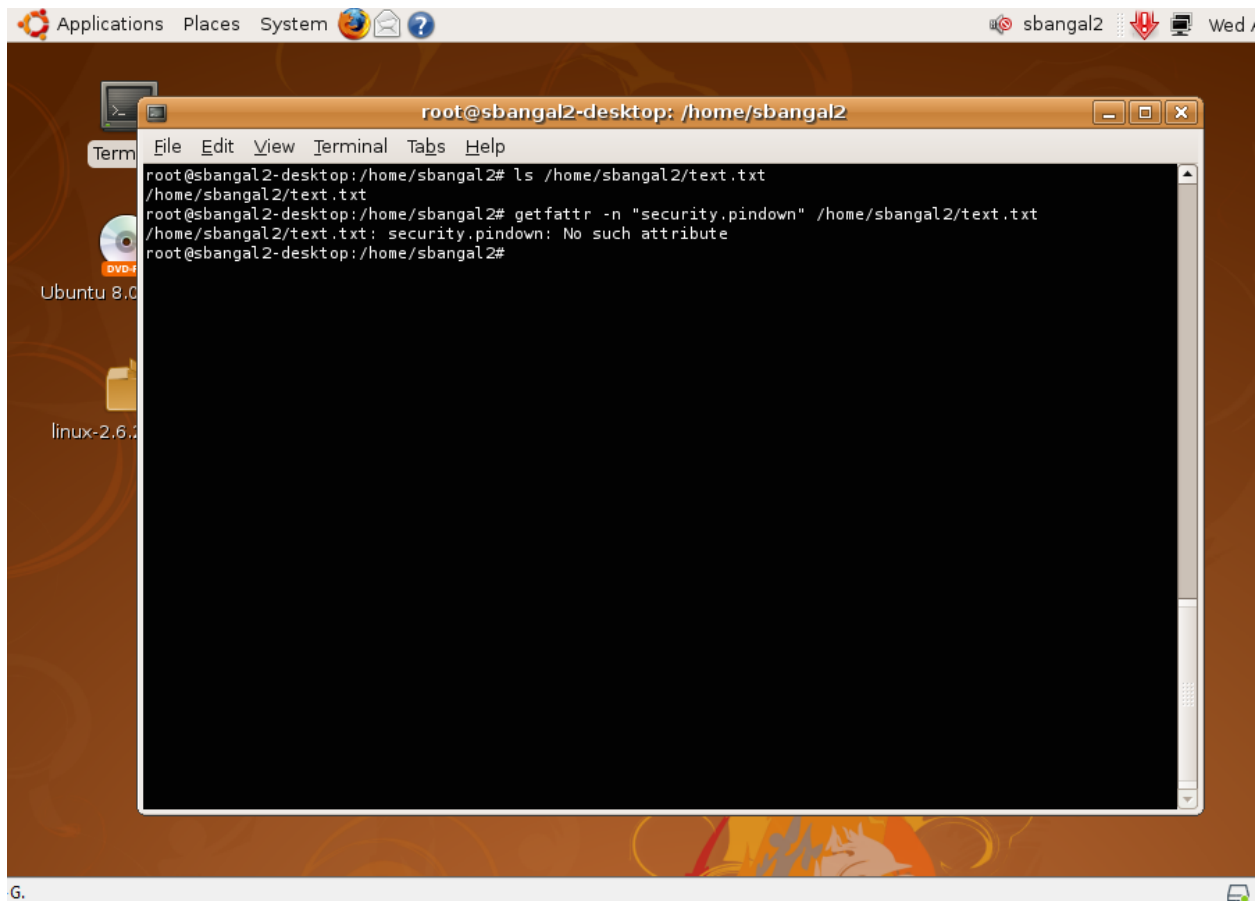
This hook is used to deallocate and clear the set security parameter allocated by **task_alloc_security()**

2. Include some screenshots of enforcement with and without Pindown.

Solution:

For **getfattr** - For each file, getfattr displays the file name, and the set of extended attribute names which are associated with that file.

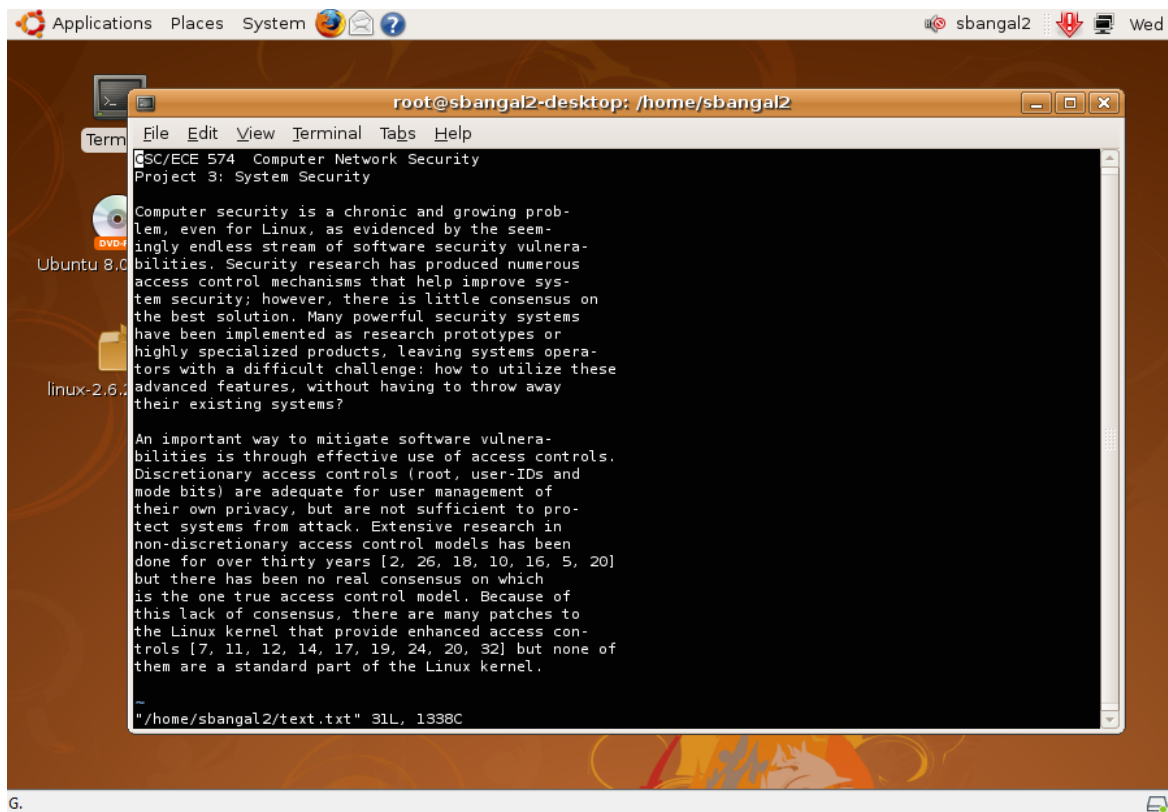
Now checking the getf attribute associated to /home/sbangel2/text.txt before the Pindown function is implemented.



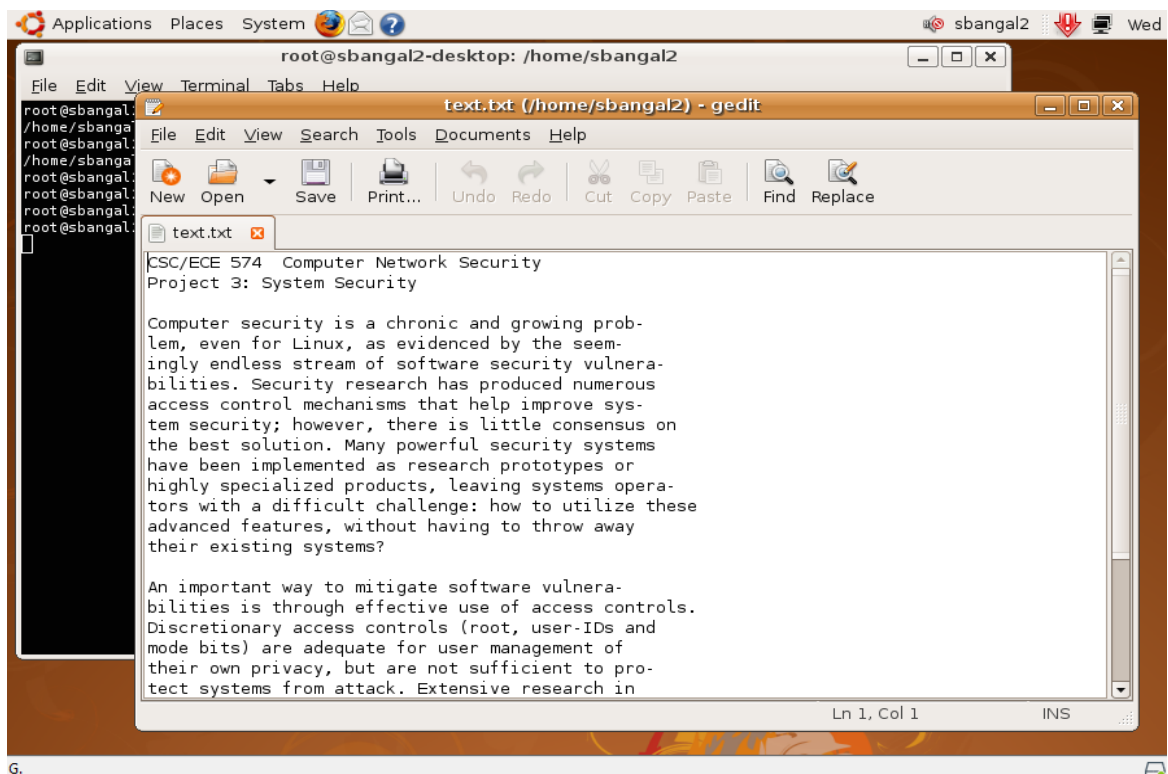
```
root@sbangel2-desktop: /home/sbangel2
File Edit View Terminal Tabs Help
root@sbangel2-desktop:/home/sbangel2# ls /home/sbangel2/text.txt
/home/sbangel2/text.txt
root@sbangel2-desktop:/home/sbangel2# getfattr -n "security.pindown" /home/sbangel2/text.txt
/home/sbangel2/text.txt; security.pindown: No such attribute
root@sbangel2-desktop:/home/sbangel2#
```

Now since the security policy are still not applied to the file /home/sbangel2/text.txt, we will be able to open using the **vim/vi** application, **gedit** application. Below are the screenshots for the same scenarios:

Opening the file (/home/sbangel2/text.txt) with vim/vi application:



Opening the file (/home/sbangel2/text.txt) with gedit application:



Compile and Loading **pindown.c** – this file is present in the file directory `/usr/src/linux-2.6.23/security/`
The content of the Makefile:

A screenshot of a Linux desktop environment. The top panel shows application icons for Firefox, LibreOffice Writer, and Nautilus, along with system status indicators like network, volume, and power. A terminal window titled "root@sbangal2-desktop: /usr/src/linux-2.6.23/security" is open, displaying a Makefile for kernel security code. The file contains rules for building modules related to keys, selinux, commoncap.o, dummy.o, inode.o, stack, built-in.o, capability.o, root_plugin.o, and pindown.o. It also includes logic for selecting between different security models based on configuration options. At the bottom of the terminal window, it says "'Makefile' 25L, 713C". On the left side of the screen, there are two vertical labels: "Ubur" near the middle and "linu" further down.

#

Makefile for the kernel security code

#

obj-\$(CONFIG_KEYS) += keys/
subdir-\$(CONFIG_SECURITY_SELINUX) += selinux

if we don't select a security model, use the default capabilities
ifneq (\$(CONFIG_SECURITY),y)
obj-y += commoncap.o
endif

Object file lists
obj-\$(CONFIG_SECURITY) += security.o dummy.o inode.o
Must precede capability.o in order to stack properly.
obj-\$(CONFIG_SECURITY_SELINUX) += selinux/built-in.o
obj-\$(CONFIG_SECURITY_CAPABILITIES) += commoncap.o capability.o
obj-\$(CONFIG_SECURITY_ROOTPLUG) += commoncap.o root_plug.o

obj-m += pindown.o

all:
 make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) modules

clean:
 make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) clean

~
~
~
~
"

"Makefile" 25L, 713C

The screenshot shows a Linux desktop with a terminal window open. The terminal window has a title bar that reads "root@sbangal2-desktop: /usr/src/linux-2.6.23/security". The terminal content shows the following commands and output:

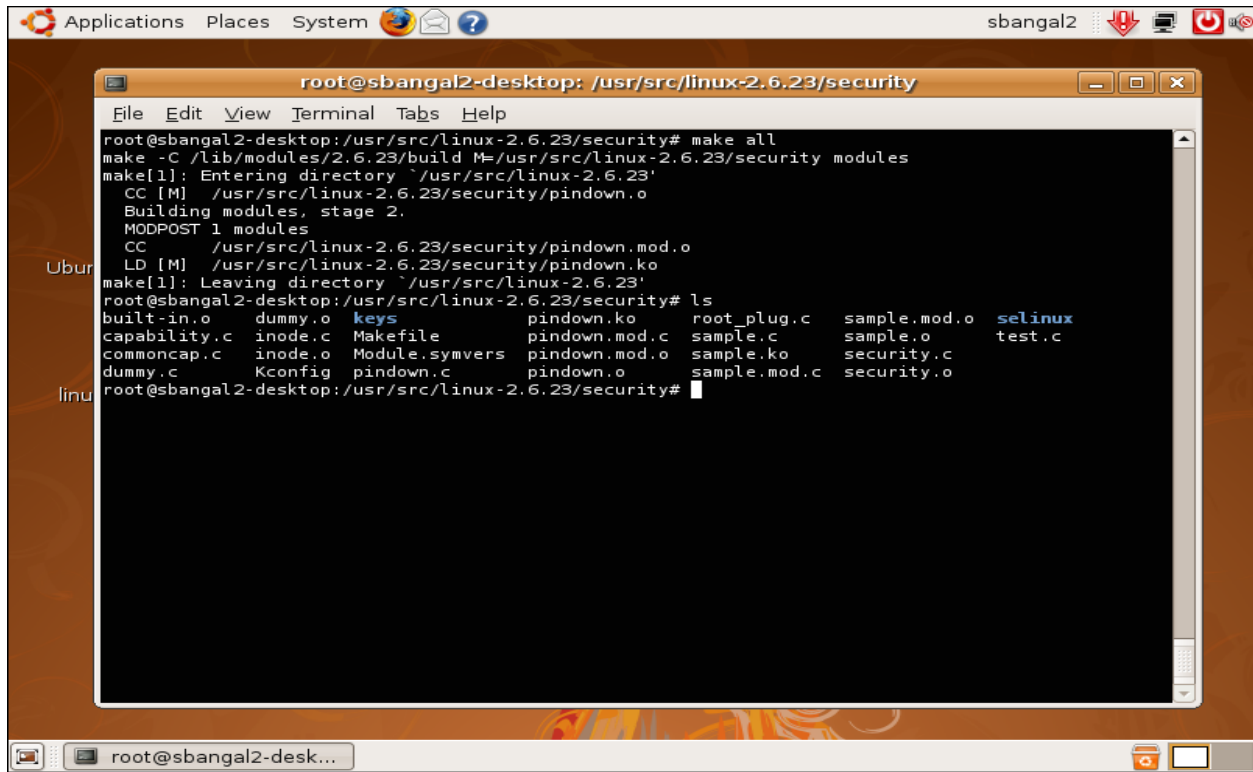
```

root@sbangal2-desktop:/usr/src/linux-2.6.23/security# vim Makefile
root@sbangal2-desktop:/usr/src/linux-2.6.23/security#
root@sbangal2-desktop:/usr/src/linux-2.6.23/security# ls
built-in.o      dummy.c      inode.o      Makefile      root_plug.c  sample.mod.c  security.c  test.c
capability.c    dummy.o      Kconfig      Module.symvers sample.c      sample.mod.o  security.o
commoncap.c     inode.c      keys         pindown.c     sample.ko    sample.o      selinux

```

The terminal window is part of a desktop environment with a taskbar at the bottom. The taskbar shows the application menu icon, a terminal icon, and the text "root@sbangal2-desk...". On the left side of the desktop, there is a vertical label "Ubu" and "linu".

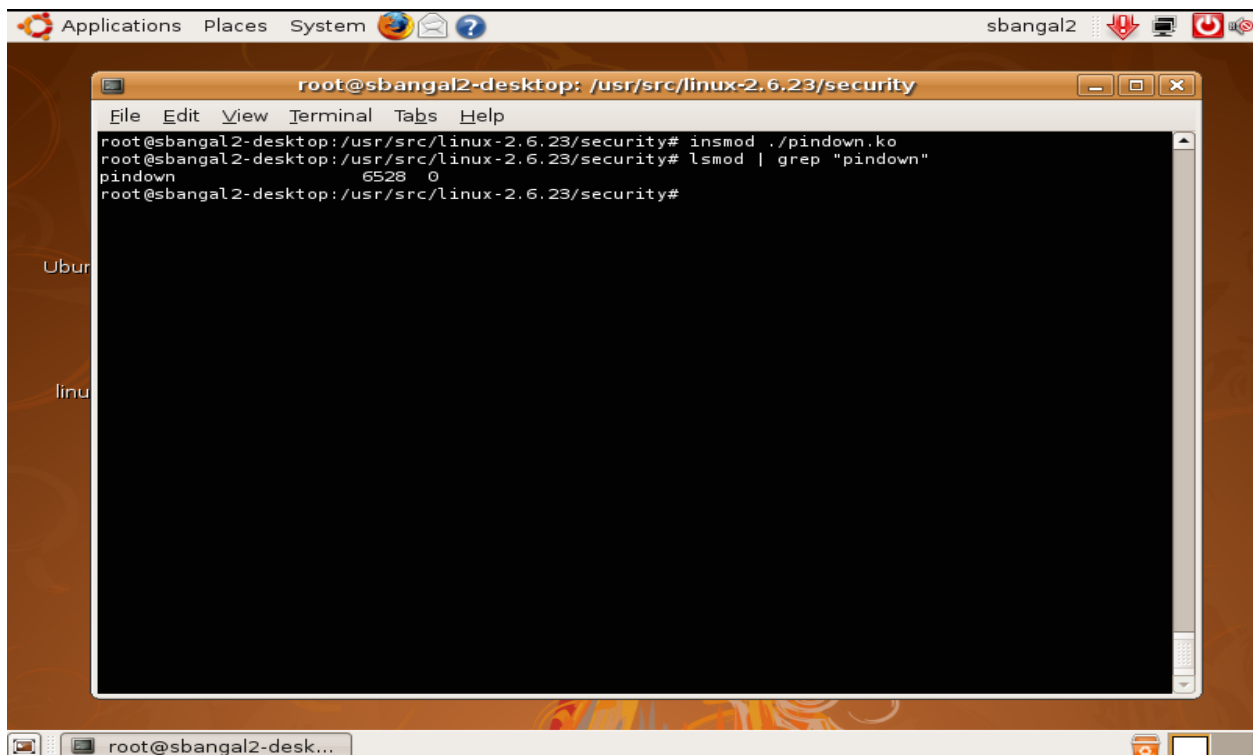
The Makefile, which enable pinDOWN.c to be compiled for the kernel. One “make all” command is run PinDOWN.ko file will be generated in the directory /usr/src/linux-2.6.23/security/



A terminal window titled "root@sbangal2-desktop: /usr/src/linux-2.6.23/security" is shown. The terminal output displays the execution of the "make all" command. It shows the compilation of "pinDOWN.ko" and a subsequent "ls" command listing the files in the directory. The files listed include "built-in.o", "dummy.o", "keys", "pinDOWN.ko", "root_plug.c", "sample.mod.o", "selinux", "capability.c", "inode.c", "Makefile", "pinDOWN.mod.c", "sample.c", "sample.o", "test.c", "commoncap.c", "inode.o", "Module.symvers", "pinDOWN.mod.o", "sample.ko", "security.c", "dummy.c", "Kconfig", "pinDOWN.c", "pinDOWN.o", "sample.mod.c", and "security.o".

```
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# make all
make -C /lib/modules/2.6.23/build M=/usr/src/linux-2.6.23/security modules
make[1]: Entering directory `/usr/src/linux-2.6.23'
CC [M] /usr/src/linux-2.6.23/security/pindown.o
Building modules, stage 2.
MODPOST 1 modules
CC /usr/src/linux-2.6.23/security/pindown.mod.o
LD [M] /usr/src/linux-2.6.23/security/pindown.ko
make[1]: Leaving directory `/usr/src/linux-2.6.23'
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# ls
built-in.o  dummy.o  keys      pindown.ko  root_plug.c  sample.mod.o  selinux
capability.c  inode.c  Makefile  pindown.mod.c  sample.c  sample.o  test.c
commoncap.c  inode.o  Module.symvers  pindown.mod.o  sample.ko  security.c
dummy.c      Kconfig  pindown.c  pindown.o  sample.mod.c  security.o
```

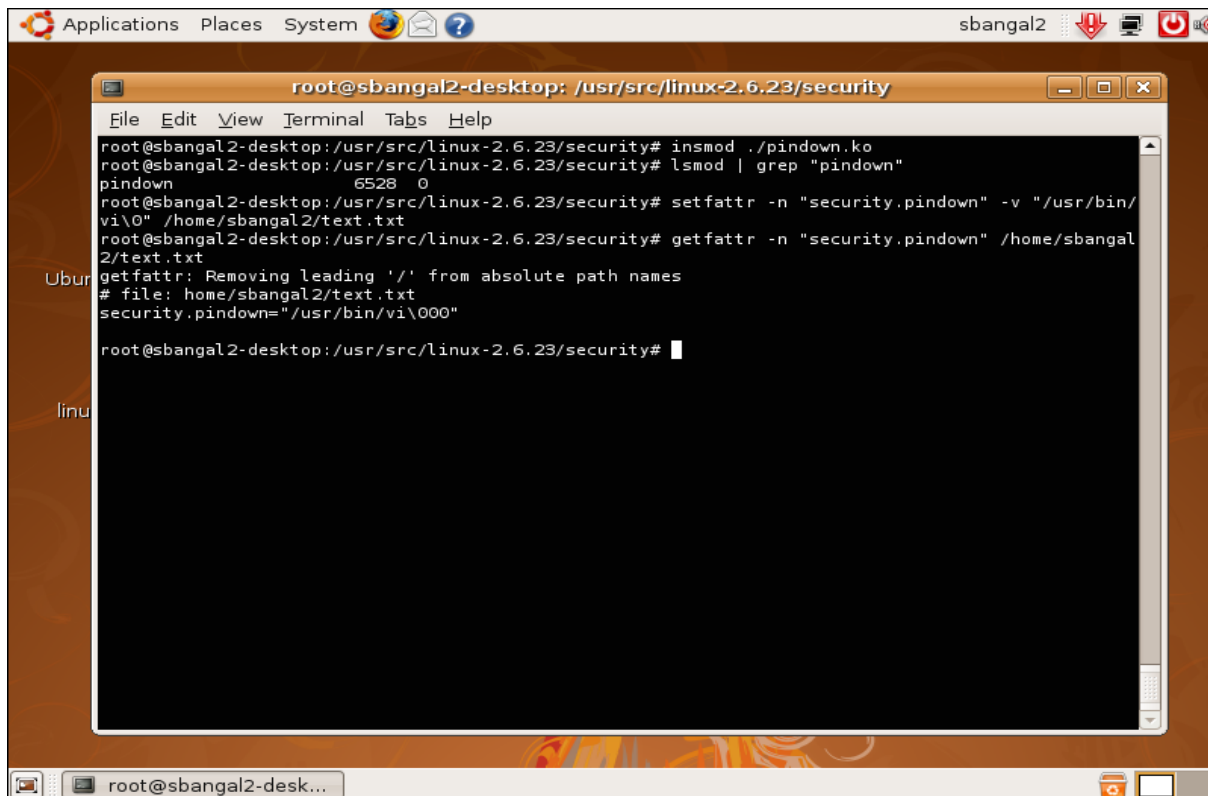
Now loading the pindown.ko file: **insmod ./pindown.ko**



A terminal window titled "root@sbangal2-desktop: /usr/src/linux-2.6.23/security" is shown. The terminal output displays the execution of the "insmod ./pindown.ko" command, followed by a "lsmod | grep 'pindown'" command. The output of the "lsmod" command shows "pindown" loaded with address "6528" and size "0".

```
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# insmod ./pindown.ko
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# lsmod | grep "pindown"
pindown                6528  0
root@sbangal2-desktop: /usr/src/linux-2.6.23/security#
```

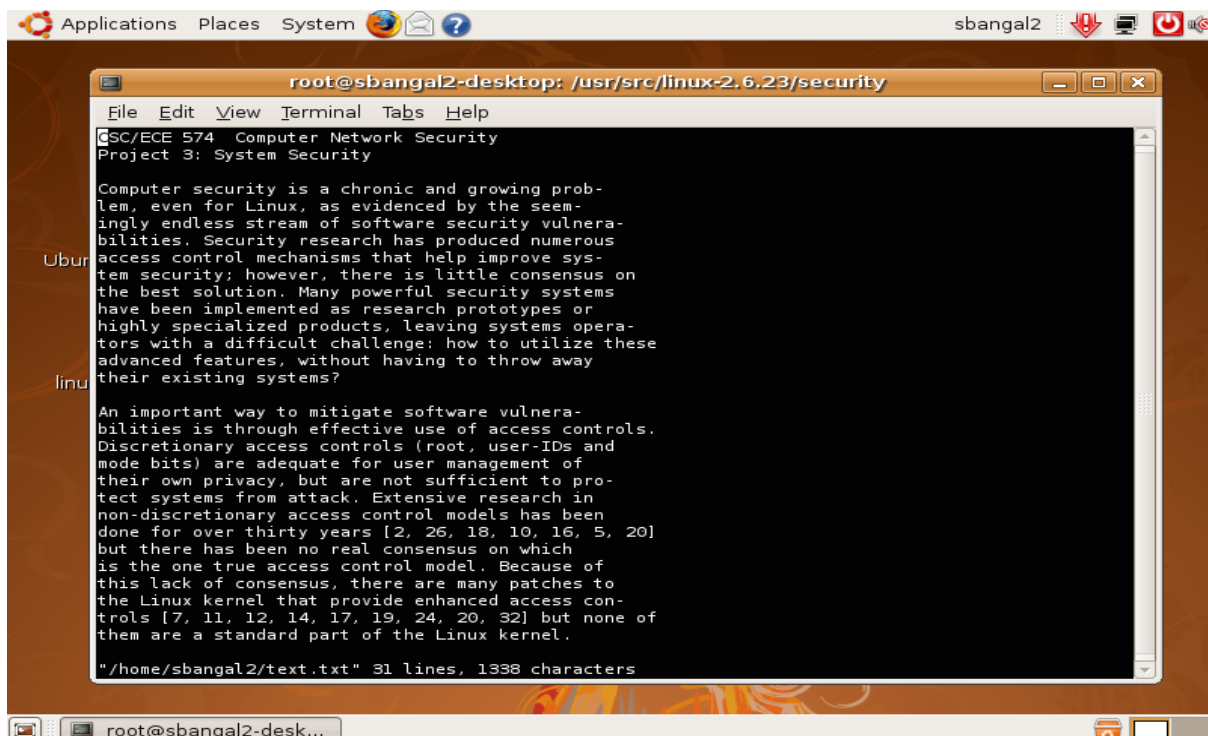
Now applying the file access control policy for the file: /home/sbanganl2/text.txt using the **setfattr** utility in Unix. Setting the access control policy such that, the file /home/sbanganl2/text.txt by only **vi** application.



```
root@sbanganl2-desktop: /usr/src/linux-2.6.23/security
File Edit View Terminal Tabs Help
root@sbanganl2-desktop:/usr/src/linux-2.6.23/security# insmod ./pindown.ko
root@sbanganl2-desktop:/usr/src/linux-2.6.23/security# lsmod | grep "pindown"
pindown                6528  0
root@sbanganl2-desktop:/usr/src/linux-2.6.23/security# setfattr -n "security.pindown" -v "/usr/bin/vi\0" /home/sbanganl2/text.txt
root@sbanganl2-desktop:/usr/src/linux-2.6.23/security# getfattr -n "security.pindown" /home/sbanganl2/text.txt
getfattr: Removing leading '/' from absolute path names
# file: home/sbanganl2/text.txt
security.pindown="/usr/bin/vi\000"

root@sbanganl2-desktop:/usr/src/linux-2.6.23/security#
```

So, the file /home/sbanganl2/text.txt can be opened only by **vi** application.



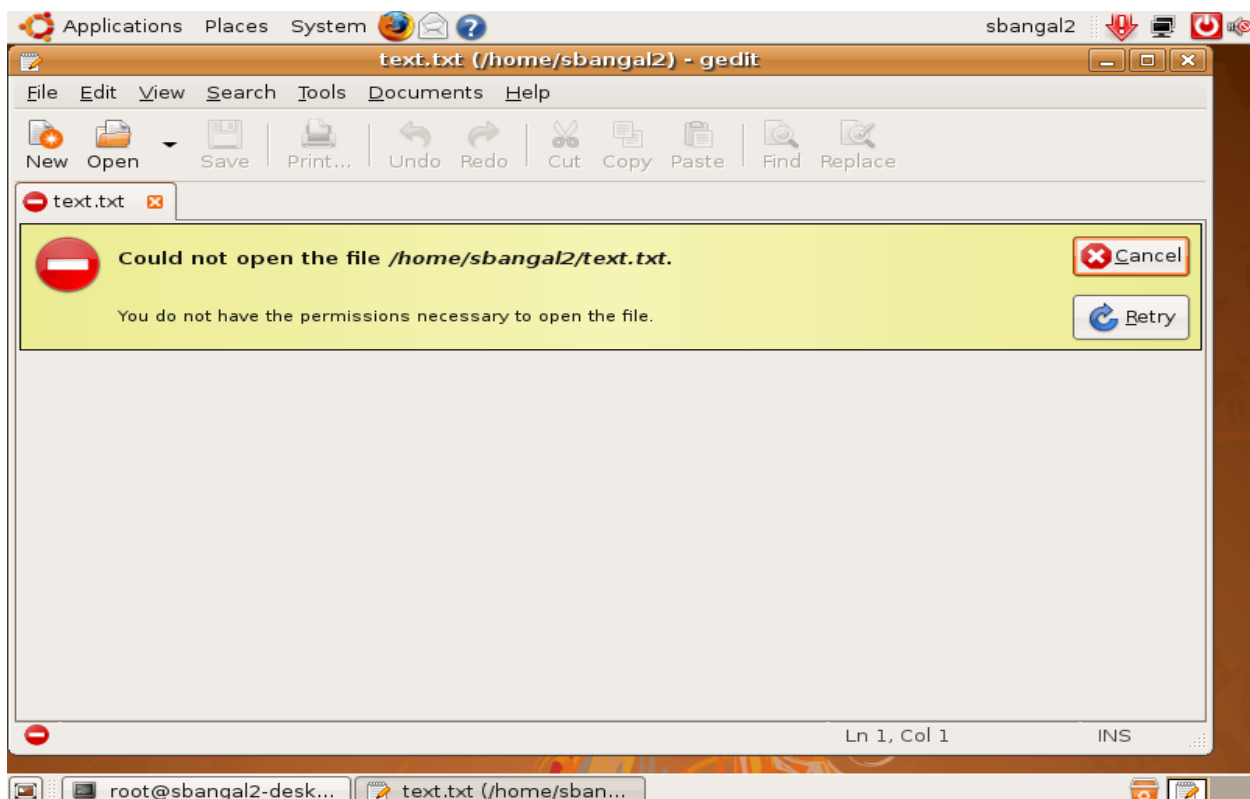
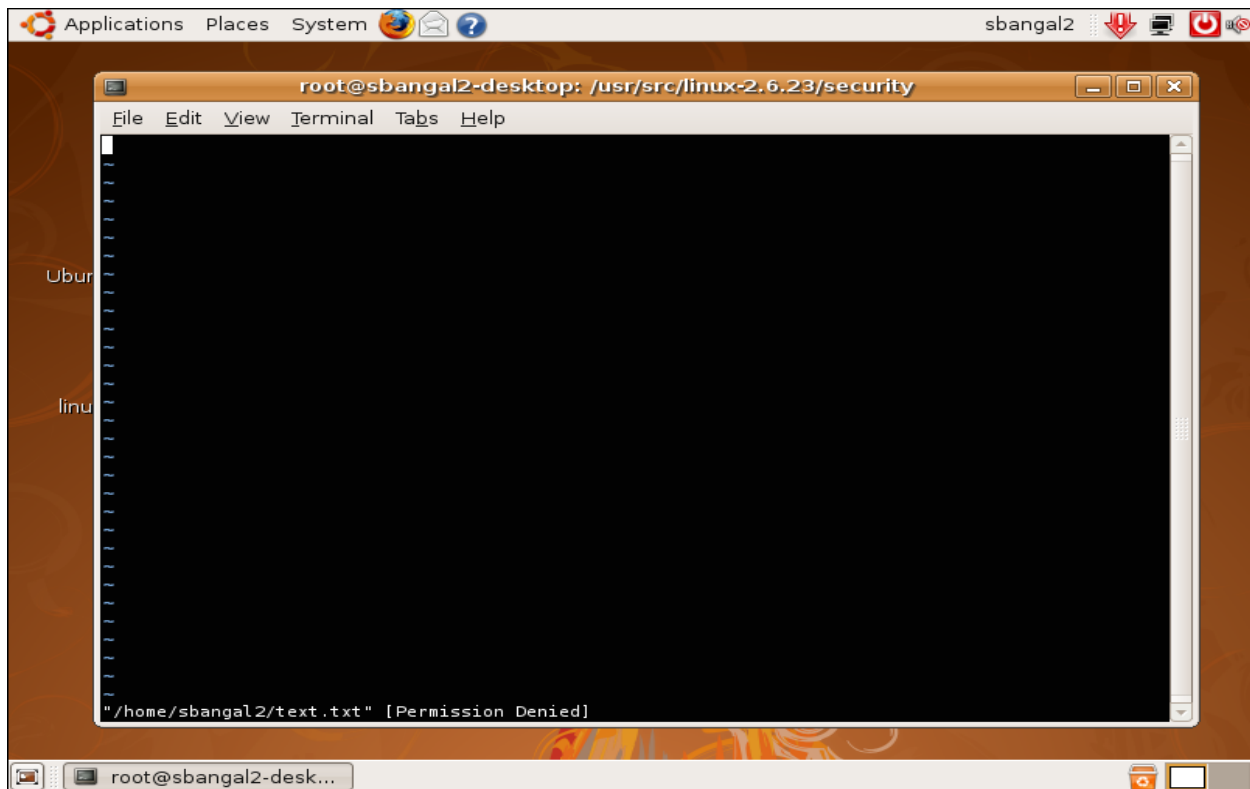
```
root@sbanganl2-desktop: /usr/src/linux-2.6.23/security
File Edit View Terminal Tabs Help
SC/ECE 574 Computer Network Security
Project 3: System Security

Computer security is a chronic and growing problem, even for Linux, as evidenced by the seemingly endless stream of software security vulnerabilities. Security research has produced numerous access control mechanisms that help improve system security; however, there is little consensus on the best solution. Many powerful security systems have been implemented as research prototypes or highly specialized products, leaving systems operators with a difficult challenge: how to utilize these advanced features, without having to throw away their existing systems?

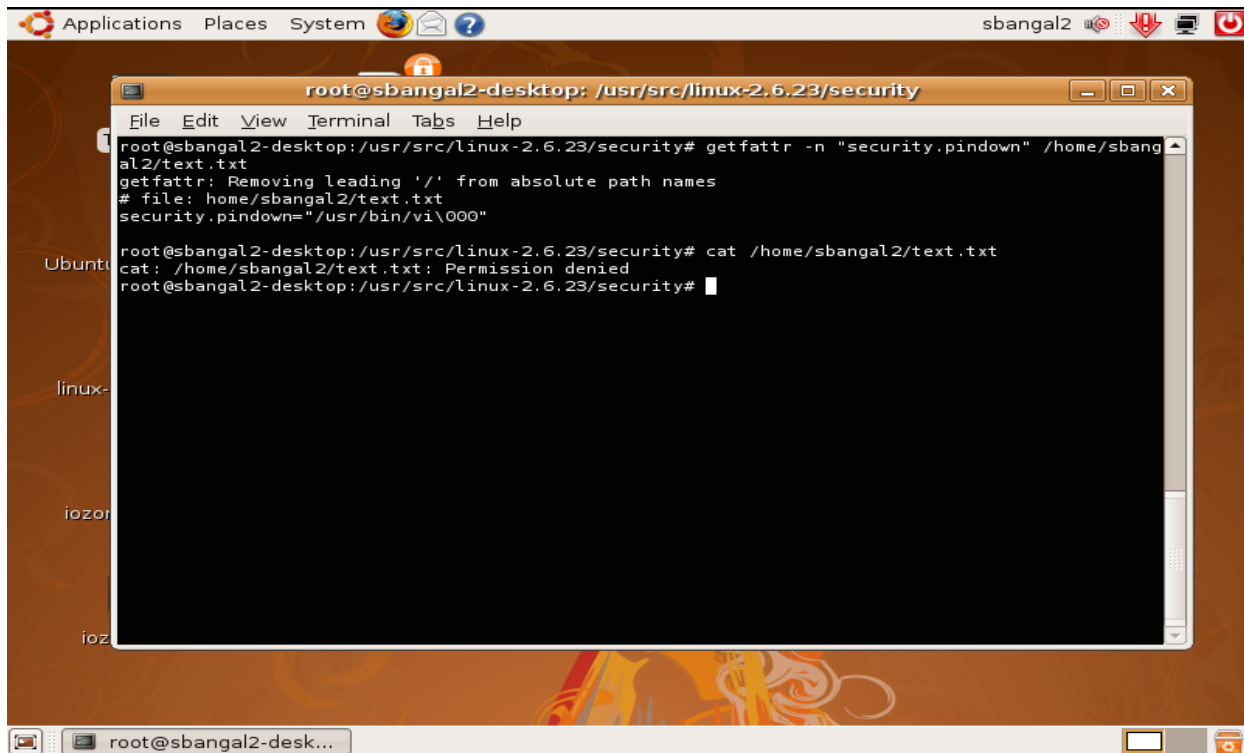
An important way to mitigate software vulnerabilities is through effective use of access controls. Discretionary access controls (root, user-IDs and mode bits) are adequate for user management of their own privacy, but are not sufficient to protect systems from attack. Extensive research in non-discretionary access control models has been done for over thirty years [2, 26, 18, 10, 16, 5, 20] but there has been no real consensus on which is the one true access control model. Because of this lack of consensus, there are many patches to the Linux kernel that provide enhanced access controls [7, 11, 12, 14, 17, 19, 24, 20, 32] but none of them are a standard part of the Linux kernel.

"/home/sbanganl2/text.txt" 31 lines, 1338 characters
```

When trying to open by **vim/gedit** application, the permission denied error will be displayed and will not be able to view the file.



The file cannot be viewed using the **cat** command also.

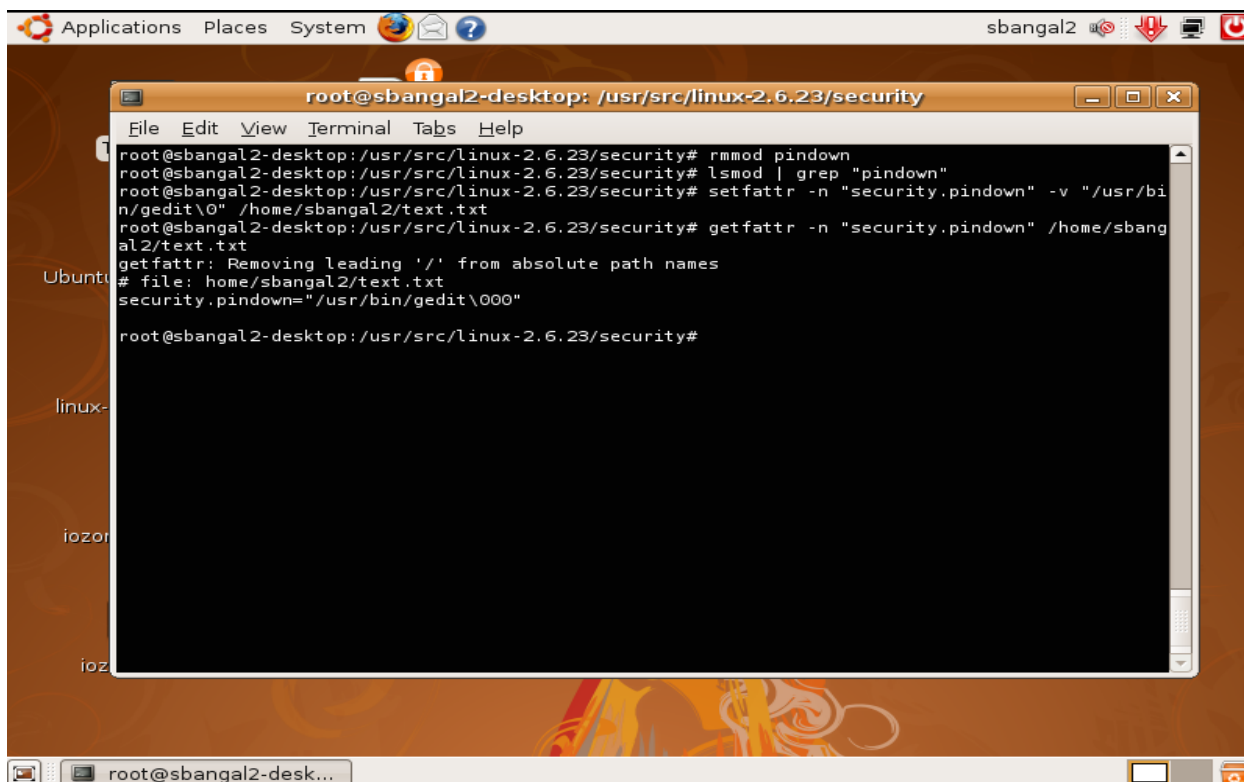


A terminal window titled 'root@sbangal2-desktop: /usr/src/linux-2.6.23/security' is open on an Ubuntu desktop. The user has run the command `getfattr -n "security.pinder" /home/sbangal2/text.txt`, which shows the SELinux policy `security.pinder="/usr/bin/vi\000"`. Then, the user runs `cat /home/sbangal2/text.txt`, which results in the error `cat: /home/sbangal2/text.txt: Permission denied`.

```
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# getfattr -n "security.pinder" /home/sbangal2/text.txt
getfattr: Removing leading '/' from absolute path names
# file: home/sbangal2/text.txt
security.pinder="/usr/bin/vi\000"

root@sbangal2-desktop: /usr/src/linux-2.6.23/security# cat /home/sbangal2/text.txt
cat: /home/sbangal2/text.txt: Permission denied
root@sbangal2-desktop: /usr/src/linux-2.6.23/security#
```

Now changing the access control policy of the file `/home/sbangal2/text.txt`, such that it can be viewed only by **gedit** application.

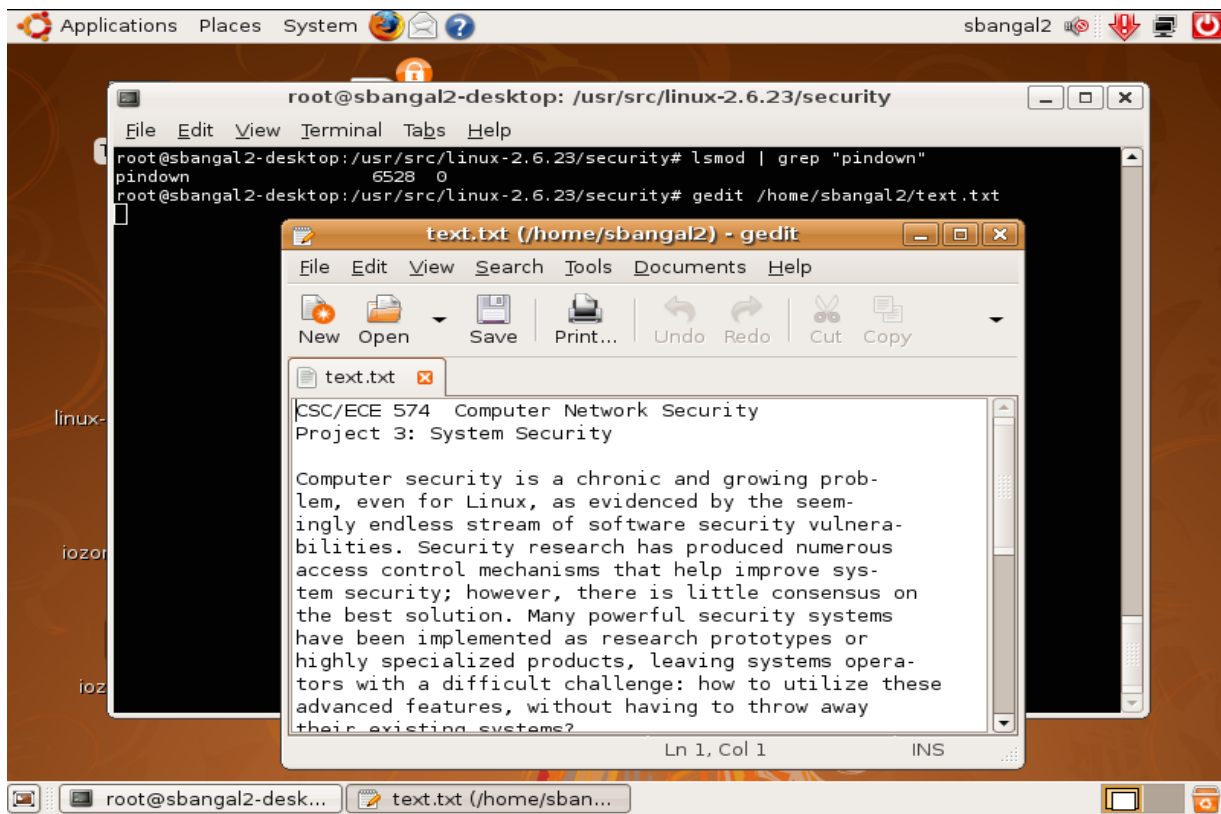


The same terminal window is shown, but now the user has removed the `pinder` policy and set the `process` policy to `gedit`. The command `getfattr -n "security.pinder" /home/sbangal2/text.txt` now shows `security.pinder="/usr/bin/gedit\000"`. The user then runs `cat /home/sbangal2/text.txt` again, and it successfully displays the file's contents.

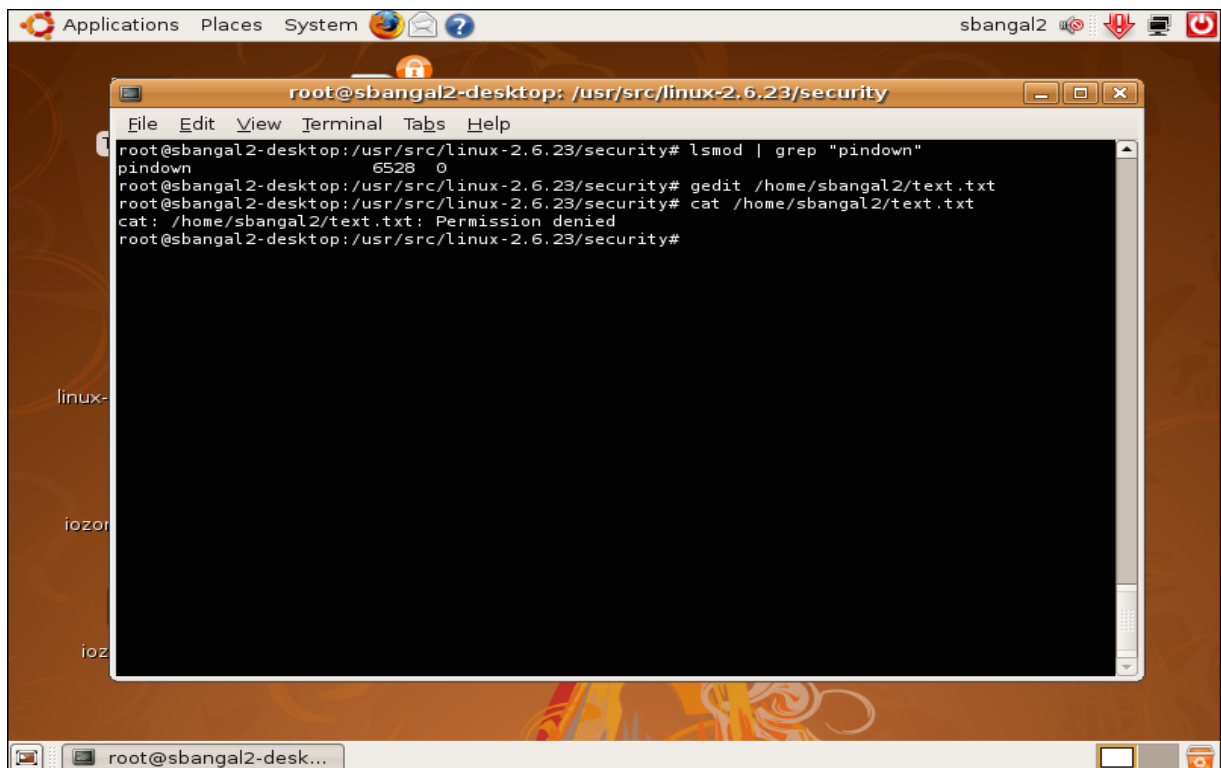
```
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# rmmod pinder
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# lsmod | grep "pinder"
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# setfattr -n "security.pinder" -v "/usr/bin/gedit\000" /home/sbangal2/text.txt
root@sbangal2-desktop: /usr/src/linux-2.6.23/security# getfattr -n "security.pinder" /home/sbangal2/text.txt
getfattr: Removing leading '/' from absolute path names
# file: home/sbangal2/text.txt
security.pinder="/usr/bin/gedit\000"

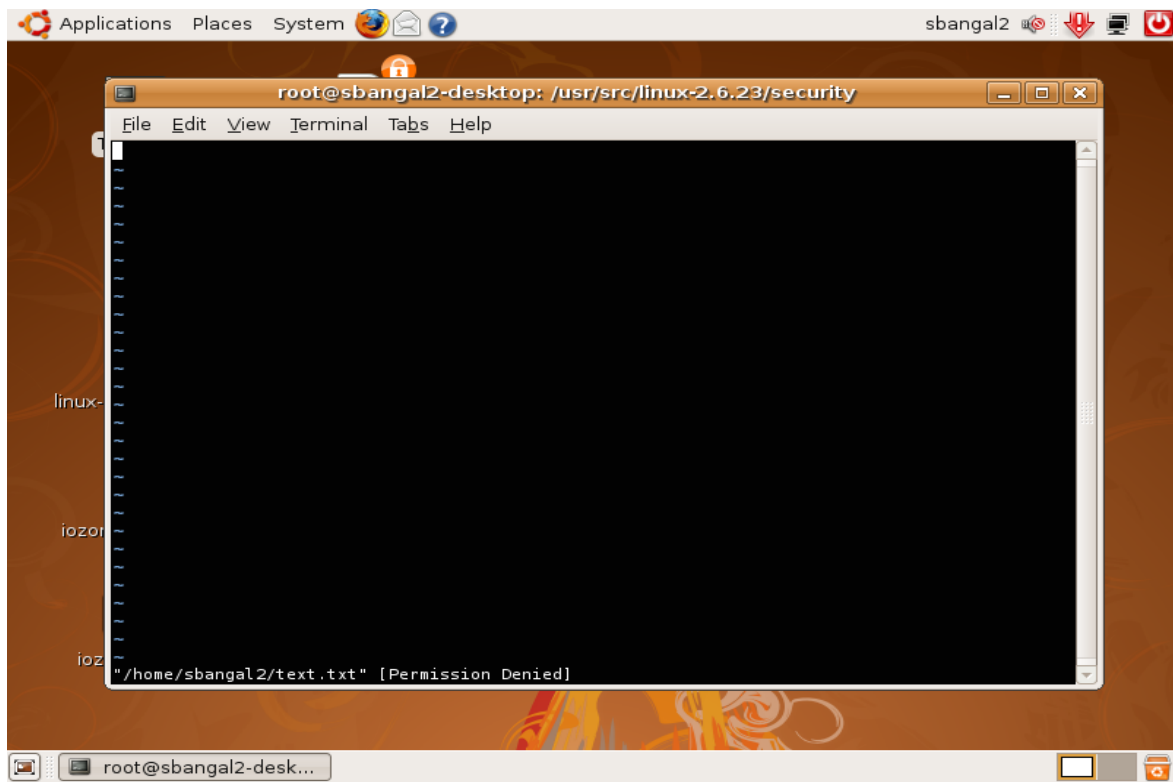
root@sbangal2-desktop: /usr/src/linux-2.6.23/security#
```

Now opening the application with **gedit** application.



Trying to open the file with other application like **vim/cat**





3. PinDOWN works well in some situations, but not others. Discuss these situations.

Solution:

Some of the drawback or limitation of the PinDOWN application are:

1. The access policy control is limited by file path:

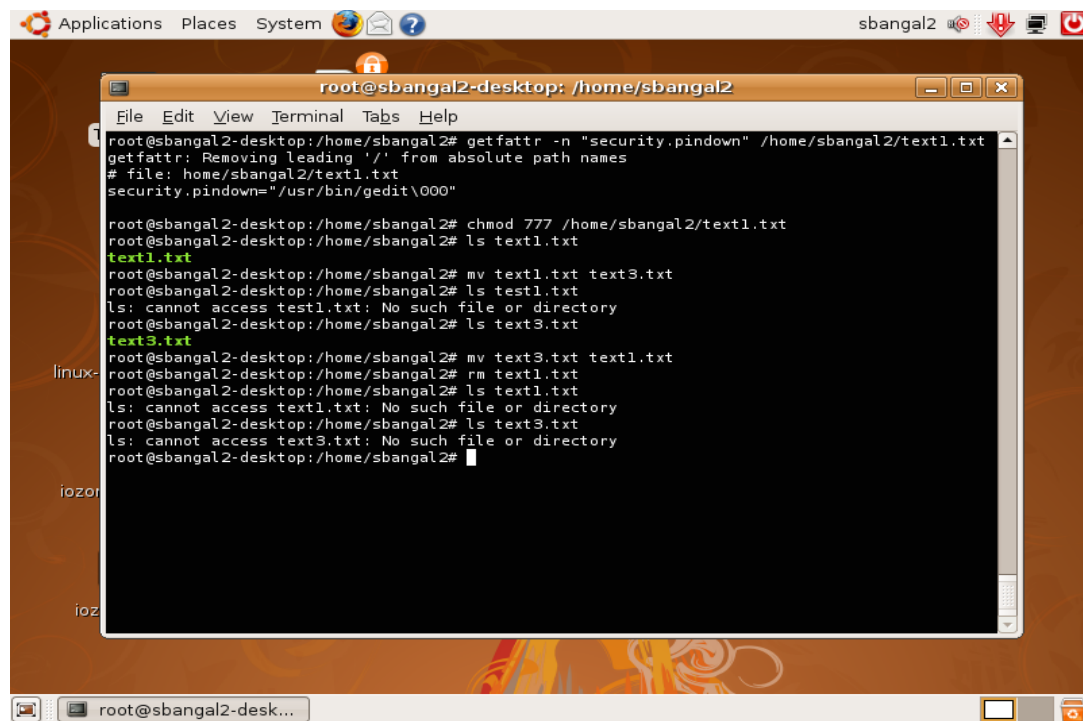
The user or role-based access control (like root user, specific username etc.) policy cannot be implemented or controlled with the help of PinDOWN application.

2. The pinDOWN application only prevent from editing by opening the file:

The pinDOWN application does not restrict the file to be

- Removed using the **rm** command.
- Moving the file to different name completely using **mv** command.
- Changing the permission of the file using **chmod** command.

This might lead to potential malicious code.



```
root@sbangal2-desktop: /home/sbangal2
File Edit View Terminal Tabs Help
root@sbangal2-desktop:/home/sbangal2# getfattr -n "security.pindown" /home/sbangal2/text1.txt
getfattr: Removing leading '/' from absolute path names
# file: home/sbangal2/text1.txt
security.pindown="/usr/bin/gedit\000"

root@sbangal2-desktop:/home/sbangal2# chmod 777 /home/sbangal2/text1.txt
root@sbangal2-desktop:/home/sbangal2# ls text1.txt
text1.txt
root@sbangal2-desktop:/home/sbangal2# mv text1.txt text3.txt
root@sbangal2-desktop:/home/sbangal2# ls text1.txt
ls: cannot access text1.txt: No such file or directory
root@sbangal2-desktop:/home/sbangal2# ls text3.txt
text3.txt
root@sbangal2-desktop:/home/sbangal2# mv text3.txt text1.txt
root@sbangal2-desktop:/home/sbangal2# rm text1.txt
root@sbangal2-desktop:/home/sbangal2# ls text1.txt
ls: cannot access text1.txt: No such file or directory
root@sbangal2-desktop:/home/sbangal2# ls text3.txt
ls: cannot access text3.txt: No such file or directory
root@sbangal2-desktop:/home/sbangal2#
```

3. The security policy will not be applied to that file when the system is restarted.

Once the system is restarted or rebooted the pinDOWN process will be terminated and the file can be accessed using any application binaries like **vim/vi/gedit/cat** etc.

insmod ./pindown.ko file needs to be executed when the system is restarted.

4. The pinDOWN application provides access control policy to the particular file and the control policy needs to be applied to each file individually. We cannot provide or add access control policy to a directory consisting of multiple files. The user will be able to list or view all the files present in a particular directory structure.

5. If the path name of the file is changed pinDOWN application fails:

Since in the pinDown application pins the file and adds access control policy based on the file path, if the path name is modified then the pinDOWN application fails.

4. Provide general observations about PinDOWN and about the project. For example, what was the most difficult part of the project. What advice would you give to future students completing the project?

In this project the main aim is to build access control system for Linux using the Linux Security Module (LSM). The application PinDOWN built is similar to PinUP application and the application uses the path of the file to apply access control policy. The major challenge in this assignment is to understand the LSM modules and complexities of the linux kernel programming. Since I have not dealt much with C programming, it was a necessary to learn the C programming language basics and understand its difference with respect to other high-level language like python. C program relies heavily on pointers unlike python, it took little more time to become more familiar with pointers usage and operations. The sample code which was provided by the instructors was helpful in understanding the code structure and how to proceed with writing the code.

PinDOWN application a simplified version which was asked to implement, used path of the file instead of the hash (used in PinUP) to apply access control policies to the file. The papers to refer which was suggested by the instructors (LSM Framework and PinUP) gave a brief idea on what needs to be implemented in the project.

Few Challenges: As mentioned above, the major challenge was to get good hang of c programming language to add on to the sample code. Bring up the virtual machine and the required setup was little time consuming since this was first time working on building custom kernel. I did face few challenges on saving kernel configuration and enabling LSM security options. The document provided by the instructors and the piazza post helped me to solve the problem. For performance analysis I tried with some tools mentioned in the link, like bonnie, iozone. But while running bonnie performance tool, the tool took too much time to report the result and it required the file size to be 2 the allocated RAM size for best result. The system did go down for some reason which I was not able to find out. Because of these issues, I decided to use iozone tool. The following link helped in installing and running the tool: <https://www.thegeekstuff.com/2011/05/iozone-examples/>

iozone is a filesystem bench marking tool for measuring various file operations. The bench marking tests file I/O performance for following operations:

Write operations: Tests the performance of writing a new file.

Re-write operations: Tests the performance of writing a file which already exists and written.

Reader Operations: Tests the performance of reading an existing file.

Re-reader operations: Tests measures the performance of reading a file that was recently read. Etc.

Commands used for installing and running the iotzone is mentioned below:

```
wget http://www.iozone.org/src/current/iozone3\_394.tar
tar xvf iozone3_394/src/current
make
make linux

Command to run the tool:
/home/sbanga12/Desktop/iozone3_394/src/current/iozone -a -g 1024m -Rb
<outputfile.wks>
```

Advice to students:

1. Follow the instructions given by the TA and use VMware workstation rather than Virtual box. I face few issues while using virtual box. VMware workstation is more user friendly.
2. It is important to go through the Pinup and LSM framework paper as reference, as it gives you good idea on what you are going to accomplish in the project.
3. Before reviewing the sample.c code and editing it, it is better to brush all the concepts in C programming especially structures, pointer, initializing and dereferencing the pointer etc.
4. Piazza post is useful to refer when you are stuck in any part of the project.

QUESTION 2: SECURITY EVALUATION.

As described above, PinDOWN is not tamperproof. In this question, we will explore the weaknesses of PinDOWN and suggest ways of addressing them.

1. Write an appropriate threat model for PinDOWN. Note that your implementation might not meet this threat model.

Solution:

- PinDown application uses only path of the file to apply control access policy. An adversary can change the path of the file by moving the file to another location. The pinDown application does not restrict the use of **mv** command to move the file to another location.
- The adversary can delete the file even though control access policy has been applied to the file using **setfattr** attribute. Deletion of important binaries can lead to system crash or malfunctioning of the application.

- The adversary can change the permission of the file using the **chmod** command even though control access policy is applied. This will restrict other application from editing/accessing the file.
- The adversary can delete the directory in which the file exists. The pinDown application cannot restrict this operation as the setfattr is applied only to the file not the directory.
- If the adversary knows the location of the location of the pinDown application, the functionality of the pinDown can be altered and replaced by malicious code. If the adversary can login to the system (as a root), they can alter the control policy (rmmod pindown).
- The PinDown application is not restricted to user-based access. Any user can access the file to which access control policy is applied.
- The pinDown application requires **insmod ./pindown.ko** command to be run each time the system is restarted or rebooted. Therefore, if the adversary somehow manages to restart or reboot the system, they can surpass the access control policies.

2. Describe how PinDOWN can be modified to meet the threat model you just described.

Solution:

Some of the suggestion on modification of the PinDOWN application to meet the threat model are:

- The application which is used to apply the access control policies to the file, should also prevent the operation like:
 - o Moving the file to another directory.
 - o Deletion of the file
 - o Changing the permission of the file.
- The PinDOWN application should be modified such that it should restrict the user/application from accessing the file based on the role assigned (Ex. Root user etc.). The access policies applied by the root user should not be visible to the other users.
- Other drawback of the PinDown application is that, the control access policies are applied only to the specific files. The application should support applying policies to the directories as well. It will prevent from deletion of the directory containing the file by the adversary.
- It would be better to apply mechanism like pinUp which uses cryptographic hash to identify the program and other important hooks that pinUp modules use.

- The PinDown module should be modified such that the insmod ./pindown.ko process should be executed as soon the system status is up and running after restart/reboot. This prevent from manually enabling the access control policies every time the system is rebooted.

QUESTION 3: PERFORMANCE EVALUATION.

In this final question, you will perform a small performance evaluation of your PinDOWN implementation. Find an appropriate file system benchmark and run it both with and without the PinDOWN module loaded. Note that since you are running in a VM and PinDOWN makes minor changes to the accessing of files, there should be negligible overhead. The purpose of this question is simply to gain experience performing a performance evaluation. You might start at this (older) page describing filesystem benchmarks. For this question, report tables or graphs that are appropriate for the benchmark that you chose.

Solution:

For analyzing the performance impact after enabling the pinDown application I have used iозone performance tool. iозone is a filesystem benchmark tool. The benchmark generates and measures a variety of file operations. For broad filesystem analysis iозone is used. The tool tests the file I/O performance by implementing following operations: Write, read, re-write, re-read, random read/write etc.

Steps followed to run the tool:

Installation: <https://www.thegeekstuff.com/2011/05/iozone-examples/>

- wget http://www.iozone.org/src/current/iozone3_394.tar
- tar xvf iozone3_394/arc/current
- make
- make linux

Command to run the iозone tool:

- Enable the pinDown application and execute the following command.
- /home/sbangan2/Desktop/iozone3_394/src/current/iozone -a -g 256m -Rb <outputfile.wks>

Some of the parameter options that can be given to the iозone tool:

- a - fully automatic mode for record sizes of 4k to 16M for file sizes of 64k to 512M.
- g - setting the maximum file size for automatic mode.
- Rb – generate the output as an excel file of given name.


```

root@sbangan2-desktop: /home/sbangan2/Desktop/iozone3_394
File Edit View Terminal Tabs Help
root@sbangan2-desktop:/home/sbangan2/Desktop# ls -lrt
total 58008
-rw-r--r-- 1 root root 1679360 2011-04-26 10:17 iozone3_394.tar
-rw-r--r-- 1 sbangan2 sbangan2 5554 2020-03-19 18:12 gnome-terminal.desktop
-rw-r--r-- 1 sbangan2 sbangan2 57404023 2020-03-19 20:12 linux-2.6.23.tar.gz
drwxr-xr-x 4 root root 4096 2020-04-01 20:20 iozone3_394
-rw-r--r-- 1 root root 50489 2020-04-01 20:36 output_with_access_policy.wks
-rw-r--r-- 1 root root 50489 2020-04-01 20:50 output_without_access_policy.wks
-rwxrwxrwx 1 root root 1338 2020-04-02 14:06 text.txt
-rw-r--r-- 1 root root 50489 2020-04-03 04:05 Outputfile_with_pindown_app.wks
-rw-r--r-- 1 root root 50489 2020-04-03 10:40 Outputfile_without_pindown_app.wks
root@sbangan2-desktop:/home/sbangan2/Desktop# cd iozone3_394
root@sbangan2-desktop:/home/sbangan2/Desktop/iozone3_394# ls -lrt
total 8
drwxr-xr-x 3 root root 4096 2020-04-01 20:20 src
drwxr-xr-x 2 root root 4096 2020-04-01 20:20 docs
root@sbangan2-desktop:/home/sbangan2/Desktop/iozone3_394# /home/sbangan2/Desktop/iozone3_394/s
rc/current/iozone -a -g 1024m -Rb Outputfile_with_access_policy.wks

```

In the excel sheet generated by the tool, table is generated as file size vs record size. From the observation from the graphs, since the pindown application adds extra overhead the rate of accessing the file will decrease when the Pindown is enabled.

1. Write Operation.

The tabular column for performance result with PinDown enabled for Write Operation.

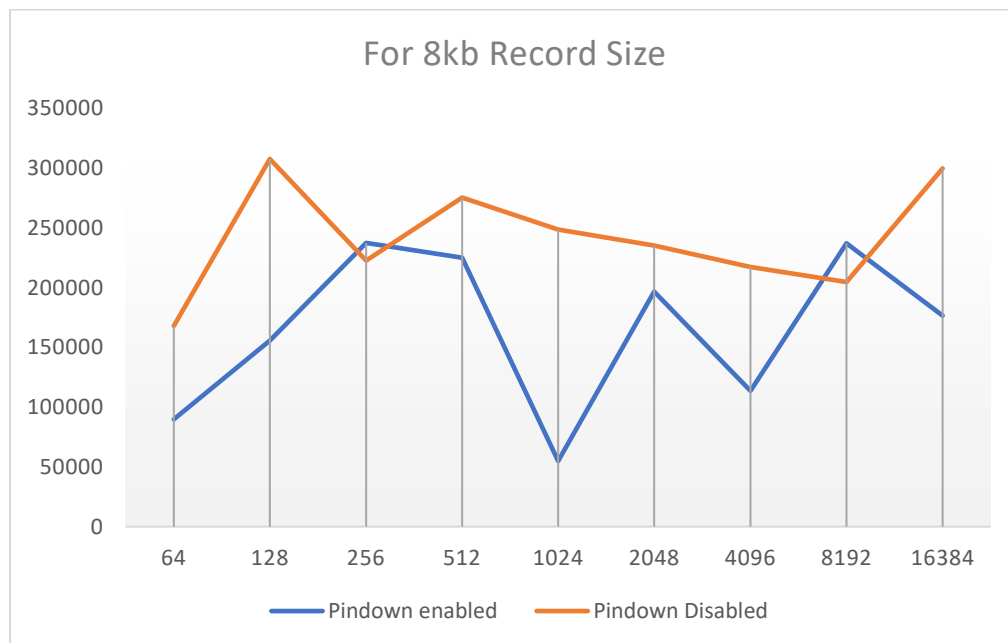
With Pindown Enabled	Record Size				
File Size (Kb)	4	8	16	32	64
64	19650	89873	12892	179219	112655
128	21241	155713	267125	27990	241528
256	167751	237458	9238	42531	22353
512	229903	225154	26517	34473	29423
1024	77629	54970	189140	271991	149949
2048	42377	196697	33120	321457	225553
4096	50625	113740	265766	240601	253435
8192	48652	237009	233430	220784	253291

16384	54611	176496	290944	331405	316500
-------	-------	--------	--------	--------	--------

The tabular column for performance result with PinDown disabled for Write Operation

Without pindown	Record Size				
FileSize in kb	4	8	16	32	64
64	224668	168002	192730	329852	275939
128	332467	307696	363535	224181	352785
256	265003	222592	238143	209314	264481
512	319813	275431	229167	270164	257528
1024	275639	248654	329676	244159	248784
2048	197720	235129	276232	333825	322265
4096	275010	217442	312959	314328	291219
8192	278715	204799	277057	263424	272965
16384	296323	299815	351994	326082	343041

Now the plotting the graph for comparison - considering the 8kb Record Size



2. Read Operation.

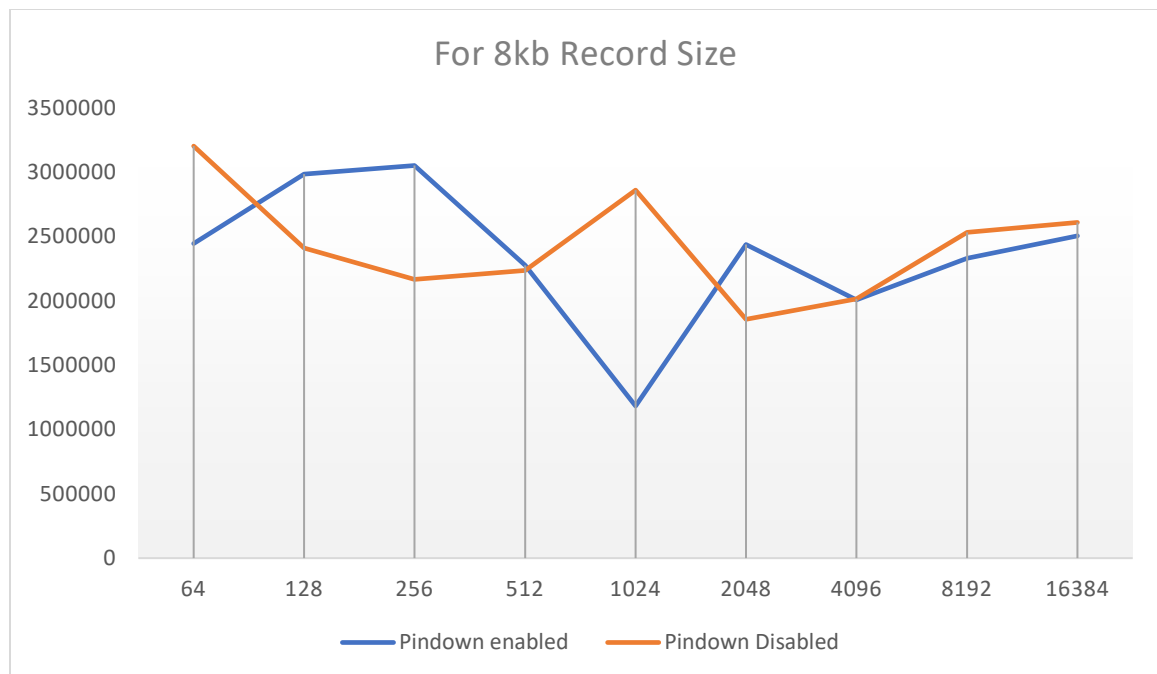
The tabular column for performance result with PinDown enabled for Read Operation.

With pindown	Record Size				
File Size	4	8	16	32	64
64	2203800	2444640	3022727	2772930	5389653
128	2511022	2985839	2608629	3277486	3380677
256	1552085	3052087	2880164	9114515	3043436
512	2381315	2275345	2815243	2891043	2725905
1024	133211	1182598	2701567	1333507	2516377
2048	1186486	2437686	2107025	2295665	2519185
4096	1711686	2007743	2276744	2622157	2155890
8192	1978244	2329978	2308842	2239469	2541061
16384	2153216	2504828	2697440	2779373	3172819

The tabular column for performance result with PinDown disabled for Read Operation.

Without Pindown	Record Size				
File Size(kb)	4	8	16	32	64
64	2203800	3203069	3738358	2444640	4018152
128	1561553	2409592	2558895	3759450	3124872
256	2413957	2165650	1766587	2665657	3770085
512	1947291	2237414	2845081	1770674	2995907
1024	1646334	2859868	2115949	2279941	2908281
2048	1505736	1856558	2148662	2959183	3146720
4096	1382818	2015752	1923018	2341919	2727047
8192	1943555	2531513	2290220	2493665	2624722
16384	2156053	2608860	2799186	2957354	3118961

Now the plotting the graph for comparison - considering the 8kb Record Size



REFERENCES AND ACKNOWLEDGEMENT:

1. "Linux Security Module Framework" - Chris Wright and Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman
2. "PinUP: Pinning User Files to Known Applications" - William Enck, Patrick McDaniel, and Trent Jaeger
3. Iozone Filesystem Benchmark - http://www.iozone.org/docs/IOzone_msword_98.pdf
4. Installation for Iozone - <https://www.thegeekstuff.com/2011/05/iozone-examples/>
5. Initially I tried performance tool with Bonnie++, but I was facing some error during execution. I would like to thank Amogh for having discussion regarding another tool Iozone which can be used for performance measurement. This discussion helped to proceed with performance testing.