# CSC/ECE 574 - Project 3
# Systems Security

Submitted by :
Prashanthi Kanniappan Murthy

---

## Write-up for 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.*

LSM enables loading security policies as kernel modules. PinDOWN LSM pins a file to the pathname of it to check the security permissions associated with it.

The code pindown.c has 4 main functions that deals with how the LSM hooks works.

It handles the different ways in which a file /process's pathname is set/checked can be accessed:
- Existing linux permission structure
- Child processes that are created by a parent process using fork()
- Processes that are created by exec()

1) pindown_inode_permission(@inode, @mask, @nd)

Works the way LSM hook inode_permission() does.Checks permission before accessing an inode. This is called when a file is opened and doesn't handle the read/write operations. PinDOWN doesn't handle read/write operations too, as it was mentioned in the project description.
The implementation of PinDOWN for this function works in the following way.
The process security information is got from "current" which will give the pathname from the application that is trying to access it.
We also get the pathname of the application that a file is pinned to. This is done by the utility function get_inode_policy which extracts the pathname set in extended attribute against "security.pindown" with the given path in the filesystem.
Both these pathnames are compared to decide whether a given application has access to a particular file or not.

2) pindown_task_alloc_security(struct task_struct * p )

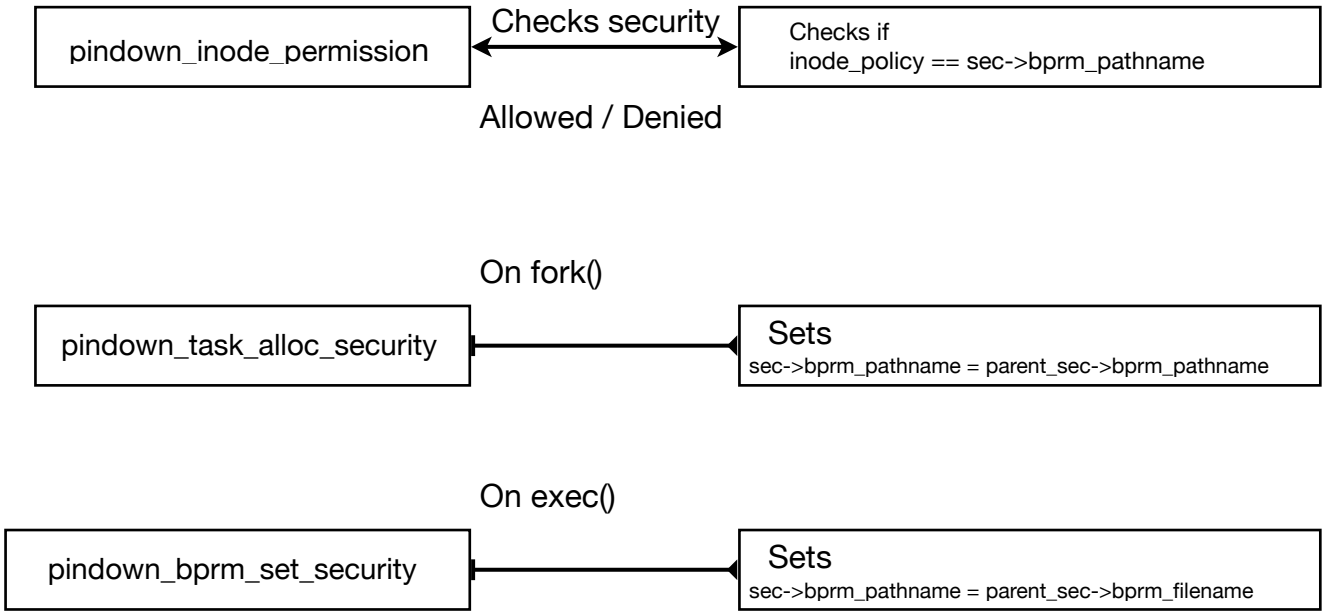
LSM hook task_alloc_security() is referred in the description. @p contains the task_struct for the child process. As mentioned in the assignment description, "*the task alloc security hook is called on fork() , copy the current process's security*

*values into the child process in task_alloc_security*". We are inheriting the parent's security field to the child.

pindown_task_free_security deallocates and clears the p->security.

3) pindown_bprm_set_security(struct linux_binprm * bprm)

LSM hook bprm_set_security() is referred in the description.This handles setting the pathname for a newly created process by exec(). The pindown security struct is inheriting the value from the binary being loaded by the kernel.



*2) Include some screenshots of enforcement with and without PinDOWN.*

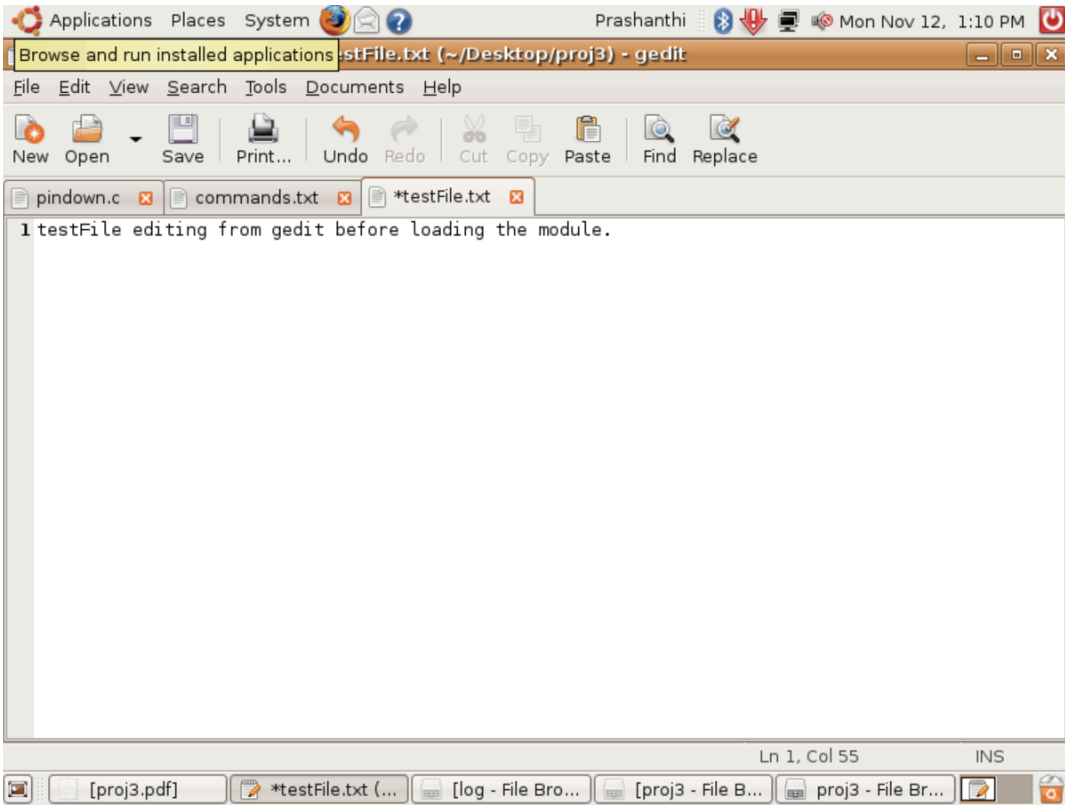



**FIG1. TESTFILE BEING EDITED BEFORE LOADING PINDOWN MODULE**

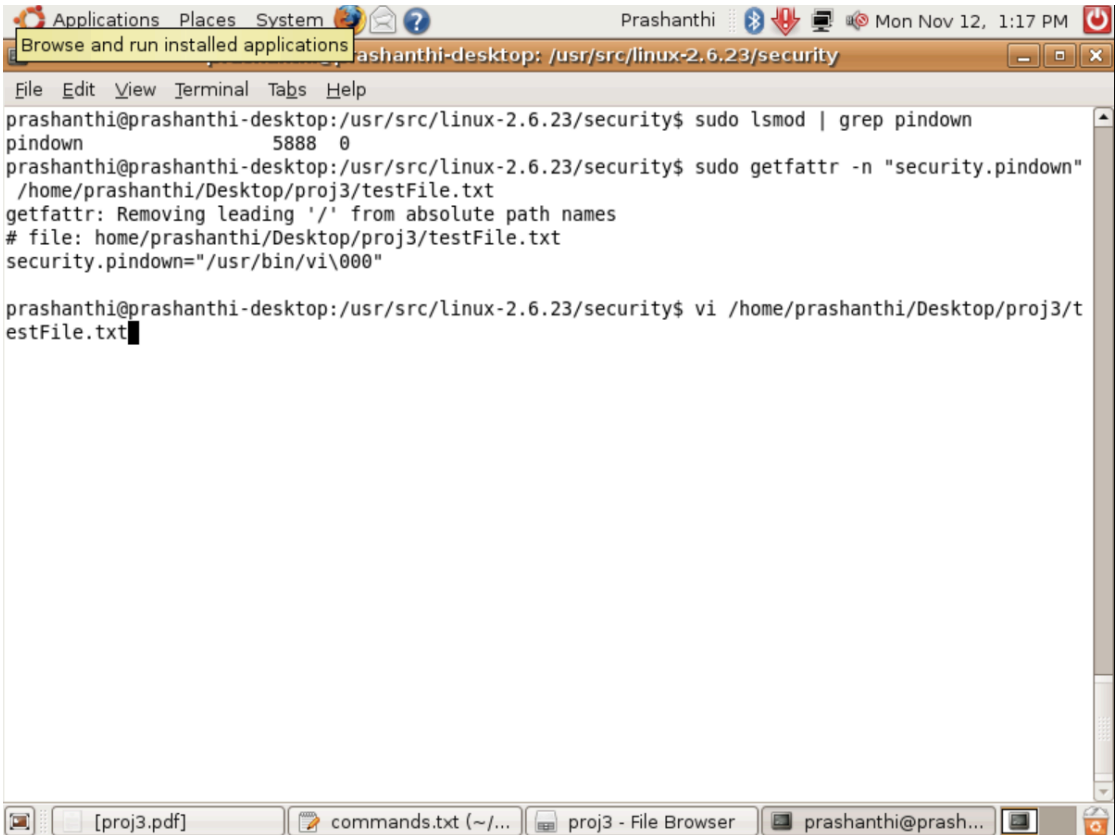



**FIG2. LOADING PINDOWN, SETFATTR TO VI, TRYING TO OPEN TESTFILE IN VI**


**FIG3. TESTFILE BEING EDITED USING VI**

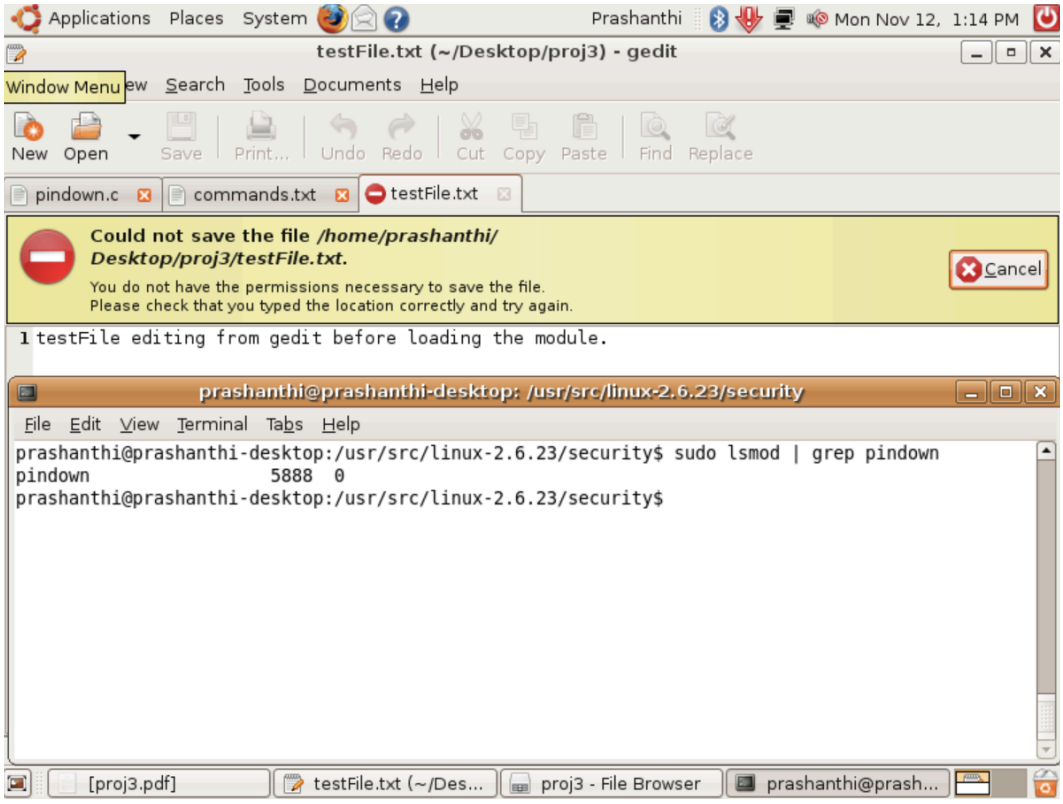



**FIG4. TRYING TO OPEN TEST FILE USING GEDIT AND ERROR THROWN FOR NO PERMISSION. SNAPSHOT ALSO SHOWS PINDOWN LOADED.**

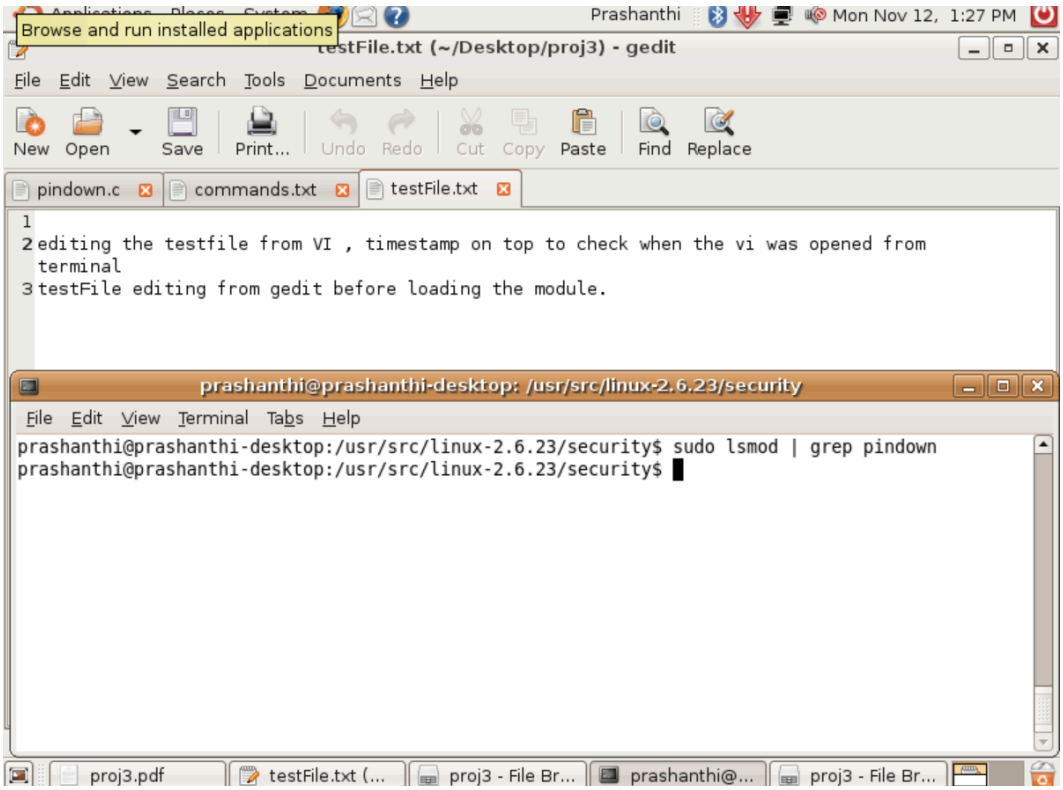



**FIG5. AFTER RMMOD, GEDIT IS ABLE TO VIEW TESTFILE WITH THE CHANGES MADE IN VI WHEN PINDOWN WAS LOADED. SCREENSHOT ALSO SHOWS PINDOWN UNLOADED.**

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

PinDOWN works good in situations where role based access are not important or may lead to bad consequences due to misuse. In situations where the user is not much aware of his security aspects and not aware of what complications he can face if he is running as root, pinning files to its applications is the best option.
On contrary to it, if a user knows about the consequences of running as a role, it is a bit of overload to pin a file to an application.If the user wants to use the file across multiple applications, for eg., opening a .txt file in a vi, attaching it in a browser to send it across..
The user here has to pin each and every application against the file.

PinDOWN restricts the access of a file in a particular path. If you move the file around, the access restrictions are gone. Cases where files are moved across for multiple operations (executing under one folder directory, opening it in particular application in one location etc.) PinDOWN doesn't work well.
But in cases where you want your file to remain only at one place, and have restrictions on it, then PinDOWN is works good in those cases.

*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?*

This assignment gives a better knowledge about how LSM s work and a deep understanding about OS and security modules in general. PinDOWN, implementing the security with only pathname makes it easier to understand how LSM works.

If your OS concepts aren't thorough, make sure you brush up the basics side by side to get better understanding.Like, how fork() and exec() works, what is an inode and how it works, how are files opened/ processes executed. For someone like me who did OS in undergrad a few years back and not using it anytime after that, basics had to be revised. It didn't take a long time as I did pretty good in undergrad OS, so revising was quite easier.If you think you are not strong, start before and revise.

The initial setup provided, if followed word by word, works perfectly fine. I did minor mistakes while typing the commands in ubuntu, because of which it took me a long time to do the setup. FOLLOW THE INITIAL SETUP INSTRUCTIONS WORD BY WORD.

The most difficult part for me was how to start with kernel C and getting basics on LSM clear.

The papers suggested in the description are a must read. To get a head start on the project code, I followed this order -> LSM basics paper -> /usr/src/linux-2.6.23/include/linux/security.h ( explanation of each hooks part ) -> PinUP paper -> PinUP open source code (although its complicated, I better understood what to write, from the PinUP code)

# 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.

To address the below 2 questions, I have taken reference from Prof.William Enck's Lecture 2 slides and from the book "Writing Secure Code" as referred in the slides.

The threat modeling process is as follows:
1. Brainstorm the known threats to the system.
2. Rank the threats by decreasing risk.
3. Choose how to respond to the threats.
4. Choose techniques to mitigate the threats.
5. Choose the appropriate technologies from the identified techniques.

Points 4,5 will answer question 2.

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

PinDOWN relies on application based security access, that the traditional role based access. The adversaries in this case could be less privileged user whose goal is to gain access to a file by trying to know what application it can have access. By doing so he can gain access to the content or modify the content as PinDOWN doesn't handle read/write access.

Since the extended attributes are set by the root user, we can assume that only the root user can use *getfattr,* to see what applications a particular file can use to gain access to it. Other adversaries could be criminals, competitors, script-kiddies who could make use of the technical errors that are left behind in the code. They can carry out some classical attacks to gain access into the file that is secured against a particular system.

Some assumptions on what an adversary cannot do are forcing the root user to set access write to a particular application to gain access.

The Trusted Computing Base (TCB) for this threat model can be the general OS tasks and processes that aren't part of the security module and are required for the OS to function. The entire hardware under the OS, we aren't expecting any hardware attacks and trust the hardware.The internet realm is not a part of TCB, we are expecting attacks from internet in the form of application updates, downloads from internet etc.

Known threats to the system are:

Kernel C 's strcpy and its issues with filling up the extra space dedicated to it with nulls. This could potential open up to classic buffer overflow attacks. An adversary who has the capability to code an executable file into the extra space available and execute it, can then make use of it to access the file he wants. Adversary can gain sudo access and set access rights for the application he wants a file to run with.

Kernel C's strcmp and its vulnerability to Timing attacks. Timing attack technique can be used by the less privileged user to "guess" what the application a particular file is pinned against, if the pathnames of the file aren't encrypted. The adversary here is capable to guess the pathname even with a polynomial running time.

The application we are pinning a file to can have defects by itself. A malicious code can be inserted into the application in the form of an update, which can gain root access and set the access to another application that the adversary wants.

Ranking them in decreasing order we have,

- application itself has a malicious code in it
- strcpy
- strcmp

To respond to these attacks,

Once we know that the application was compromised, revert the access that it had immediately to avoid further attacks

Delete the executable available in the extra buffer.

When you realize the application that has access was guessed correctly, revert access to the application or move the file to another safe folder with higher access control.

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

Techniques to mitigate the threats are :

- PinUP 's way of comparing the function with a hash, is the best way to mitigate this.
- Instead of using built-in functions strcpy, strcmp - compare/copy the strings character by character and carefully handling the end null pointer.

# Question 3: Performance Evaluation

For filesystem performance, Bonnie tool was used to get the values as mentioned in the table. [ Bonnie ]
The reports were ran on Ubuntu 8.04 LTS , kernel version 2.6.23. The host OS running on 1.8 GHz Intel Core i5.

The tabular representation is same for both before loading PinDOWN and after loading PinDOWN.

| Column | Description |
|---|---|
| **A** | File size in Mb |
| **B** | Output rate in K per second, when writing the file by doing (A) million *putc()* macro invocations |
| **C** | Consumption of CPU s time, when writing the file by doing (A) million *putc()* macro invocations |
| **D** | Output rate in K per second, when writing the file by doing (A) million putc() macro invocations, when writing the (A) size file using efficient block writes |
| **E** | Consumption of CPU s time, when writing the file by doing (A) million putc() macro invocations, when writing the (A) size file using efficient block writes |
| **F** | Ability to cover in K per second, While running through the (A)Mb file just creating, changing each block, and rewriting |
| **G** | Consumption of CPU s time, While running through the (A)Mb file just creating, changing each block, and rewriting |
| **H** | Input rate in K per second,While reading the file using (A) million *getc()* macro invocations |
| **I** | Consumption of CPU s time, While reading the file using (A) million *getc()* macro invocations |
| **J** | Input rate in K per second, While reading the file using efficient block reads |
| **K** | Consumption of CPU s time, While reading the file using efficient block reads |
| **L** | Effective seek rate in seeks per second.Bonnie created 4 child processes, and had them execute 4000 seeks to random locations in the file. On 10% of these seeks, they changed the block that they had read and re-wrote it. |
| **M** | Consumption of CPU s time, during the seek process. |

Data before loading PinDOWN module:

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| **10** | 79937 | 99.9 | 1389793 | 108.6 | 1290973 | 100.9 |
| **50** | 59457 | 98.0 | 908623 | 71.0 | 554935 | 56.4 |
| **100** | 77360 | 93.7 | 242730 | 79.7 | 699864 | 46.5 |
| **250** | 12143 | 98.3 | 221149 | 62.9 | 237309 | 45.2 |
| **500** | 13490 | 97.5 | 64100 | 10.1 | 317852 | 48.2 |
| **750** | 17621 | 80.3 | 4218 | 0.7 | 221718 | 29.8 |
| **1000** | 29625 | 94.3 | 77113 | 13.9 | 131796 | 17.0 |

| A | H | I | J | K | L | M |
|---|---|---|---|---|---|---|
| **10** | 89192 | 101.0 | 4036263 | 0 | 108292.5 | 75.8 |
| **50** | 86994 | 99.9 | 3782226 | 88.7 | 162265.2 | 97.4 |
| **100** | 83629 | 89.8 | 3943618 | 92.4 | 170017.4 | 85.0 |
| **250** | 85092 | 97.9 | 3905773 | 91.5 | 156678.4 | 78.3 |
| **500** | 82181 | 99.3 | 4577109 | 96.6 | 178667.1 | 89.3 |
| **750** | 56499 | 98.4 | 4398826 | 98.5 | 93222.7 | 46.6 |
| **1000** | 87469 | 99.2 | 5303144 | 99.4 | 215296.8 | 86.1 |

Data after loading PinDOWN Module:

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| **10** | 76582 | 101.7 | 1259067 | 98.4 | 1643923 | 64.2 |
| **50** | 74979 | 99.0 | 817408 | 89.4 | 381250 | 44.7 |
| **100** | 78952 | 96.5 | 454900 | 69.3 | 566193 | 59.7 |
| **250** | 76576 | 94.6 | 115717 | 13.9 | 275034 | 26.2 |
| **500** | 57758 | 95.0 | 141168 | 16.9 | 303862 | 30.4 |
| **750** | 63866 | 90.4 | 48539 | 6.8 | 440380 | 35.3 |
| **1000** | 66908 | 94.3 | 4755 | 1.4 | 142543 | 16.4 |

| A | H | I | J | K | L | M |
|---|---|---|---|---|---|---|
| **10** | 89643 | 98.1 | 3248730 | 126.9 | 65941.3 | 72.5 |
| **50** | 73660 | 97.8 | 3707726 | 115.9 | 145831.1 | 102.1 |
| **100** | 80355 | 98.6 | 3581922 | 97.9 | 139909.1 | 83.9 |
| **250** | 66682 | 98.3 | 3147824 | 93.5 | 171071.8 | 102.6 |
| **500** | 71741 | 99.3 | 4150723 | 94.0 | 183891.1 | 91.9 |
| **750** | 90735 | 99.8 | 4424753 | 96.8 | 193283.4 | 116.0 |
| **1000** | 68463 | 93.1 | 4879048 | 97.2 | 164473.7 | 82.2 |

As it can be seen, there is not much difference in the performance, as PinDOWN is light and doesn't do much operations that causes the overhead. The difference accounts to few thousand K/sec. Even the difference that is shown in the graph maybe due to the host machine's performance.

A more detailed report was generated using IOzone, which is available in the below links. It also covers random read and write benchmarks, which was missing in Bonnie.

Benchmarks without PinDOWN

Benchmarks with PinDOWN

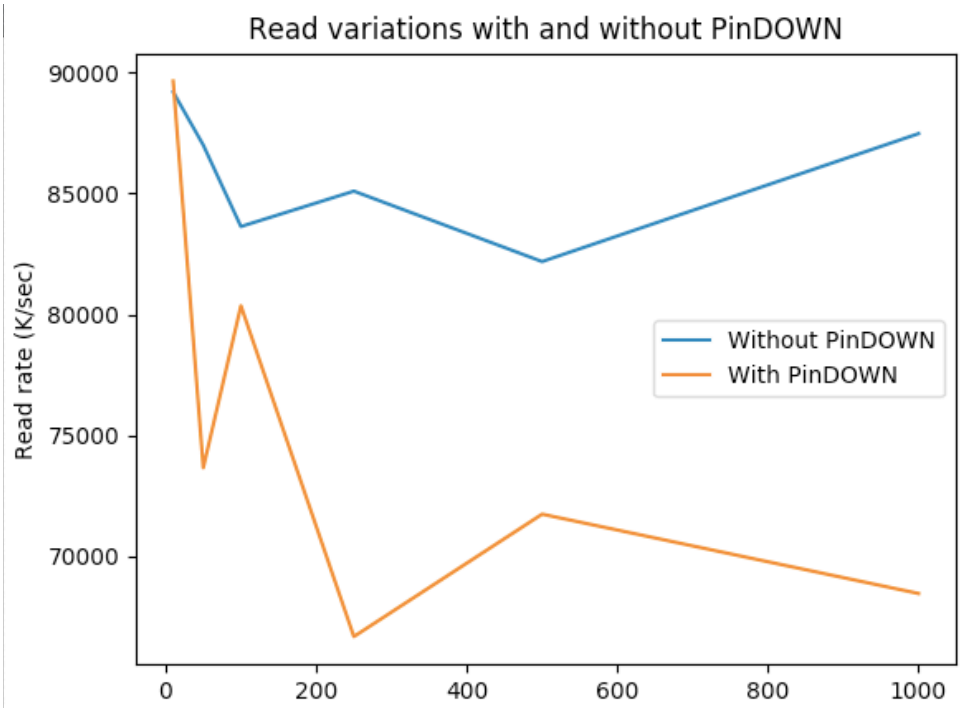



**FIG6. GRAPH GENERATED WITH PYTHON'S MATPLOTLIB USING DATA FROM BONNIE FOR SEQUENTIAL READS**

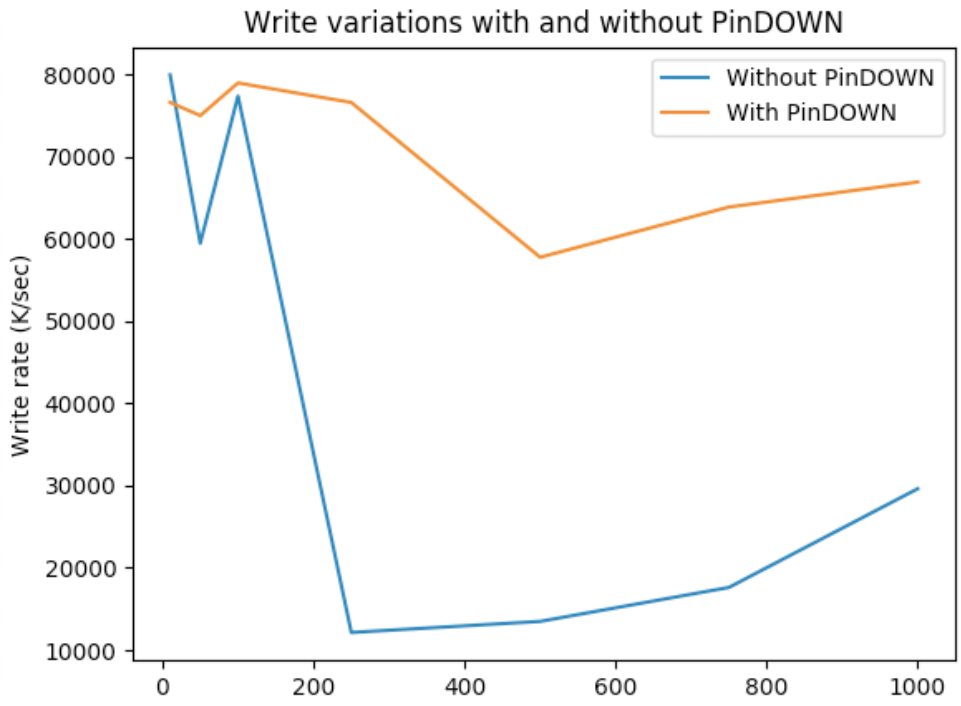



**FIG7. GRAPH GENERATED WITH PYTHON'S MATPLOTLIB USING DATA FROM BONNIE FOR SEQUENTIAL WRITES**

# Acknowledgements

I would like to thank the following people who helped me in due course of the assignment.

1) Juhi Madhwani, for motivating and directing me when I got struck, helping me to finish the project on time.
2) Suraj Siddharudh, for helping me understand threat modeling in detail .