Compile:

```
make all target=ia32
```

Execution:
Instrumented program should be present in a folder inside pin/source/tools/
e.g **pin/source/tools/myprograms/bbcount.cpp**
Pin version: Pin 3.13

pwd should be **pin/source/tools/myprograms**

```
../../../pin -t obj-ia32/pintoolName.so -- instrumentedPgm_and_arguments
cat pintoolName.out
```

Alternate:
If only 1 argument, pls use **./pintool_md toolname instrumented_pgm arg1**
To add more arguments pls add $i to the pintool_md script where i is the argument number
Ex: ./pintool_md bbcount ls

Warmup:

```
../../../pin -t obj-ia32/bbcount.so --  ls
cat bbcount.out
```

```
../../../pin -t obj-ia32/controlflow.so -- ls
cat controlflow.out
```

```
../../../pin -t obj-ia32/malloccount.so -- ls
cat malloccount.out
```

Remark: All the programs are implemented to work for multithreading programs like gedit.
Not working: For processes which fork childs like firefox, we need to instrument child processes using **PIN_AddFollowChildProcessFunction()**

Sample output:

```
sekar@ubuntu:~/Desktop/pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/final$ ./pintool_md bbcount ls
bbcount1.out  btrace1.out  controlflow1.out  log.txt         malloccount1.out  obj-ia32        stackpivot.out
bbcount.cpp   btrace.cpp   controlflow.cpp   makefile        malloccount.cpp   pintool_md
bbcount.out   btrace.out   controlflow.out   makefile.rules  malloccount.out   stackpivot.cpp
Basic Block Count 121299
sekar@ubuntu:~/Desktop/pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/final$ ./pintool_md malloccount ls
bbcount1.out  btrace1.out  controlflow1.out  log.txt         malloccount1.out  obj-ia32        stackpivot.out
bbcount.cpp   btrace.cpp   controlflow.cpp   makefile        malloccount.cpp   pintool_md
bbcount.out   btrace.out   controlflow.out   makefile.rules  malloccount.out   stackpivot.cpp
malloc count: 102
malloc size: 63050
sekar@ubuntu:~/Desktop/pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/final$ ./pintool_md controlflow ls
bbcount1.out  btrace1.out  controlflow1.out  log.txt         malloccount1.out  obj-ia32        stackpivot.out
bbcount.cpp   btrace.cpp   controlflow.cpp   makefile        malloccount.cpp   pintool_md
bbcount.out   btrace.out   controlflow.out   makefile.rules  malloccount.out   stackpivot.cpp
Direct Count 107704
InDirect Count 9891
Remaining Count 3704
```

Security application 1 (btrace):

../../../pin -t obj-ia32/btrace.so -- ls
cat btrace.out

../../../pin -t obj-ia32/btrace.so -- ./helloworld
cat btrace.out

```
sekar@ubuntu:~/Desktop/pin-3.13-98189-g60a6ef199-gcc-linux/source/tools/final$ ./pintool_md btrace ls
bbcount1.out  btrace1.out  controlflow1.out  log.txt         malloccount1.out  obj-ia32        stackpivot.out
bbcount.cpp   btrace.cpp   controlflow.cpp   makefile        malloccount.cpp   pintool_md
bbcount.out   btrace.out   controlflow.out   makefile.rules  malloccount.out   stackpivot.cpp
brk(NULL) = 0x08f5e000
access("/etc/ld.so.nohwcap", F_OK) = -1  ENOENT (No such file or directory)
mmap2(NULL , 4096 , PROT_READ|PROT_WRITE , MAP_PRIVATE|MAP_ANONYMOUS , -1 , 0x0) = 0xad6b8000
access("/etc/ld.so.preload", R_OK) = -1  ENOENT (No such file or directory)
open("/etc/ld.so.cache",O_RDONLY | O_CLOEXEC, 0) = 3
fstat64(3 , (struct address)0xbf9906e0) = 0
mmap2(NULL , 89959 , PROT_READ , MAP_PRIVATE , 3 , 0x0) = 0xad633000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1  ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libselinux.so.1",O_RDONLY | O_CLOEXEC, 0) = 3
read(3, "", 512) = 512
fstat64(3 , (struct address)0xbf990720) = 0
mmap2(NULL , 154580 , PROT_READ|PROT_EXEC , MAP_PRIVATE|MAP_DENYWRITE , 3 , 0x0) = 0xad5d8000
mprotect(0xad5fa000 , 4096 , PROT_NONE) = 0
mmap2(0xad5fb000 , 8192 , PROT_READ|PROT_WRITE , MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE , 3 , 0x22) = 0xad5fb000
mmap2(0xad5fd000 , 3028 , PROT_READ|PROT_WRITE , MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS , -1 , 0x0) = 0xad5fd000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1  ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6",O_RDONLY | O_CLOEXEC, 0) = 3
read(3, "◆◆◆[18]t◆◆", 512) = 512
fstat64(3 , (struct address)0xbf990700) = 0
mmap2(NULL , 1792540 , PROT_READ|PROT_EXEC , MAP_PRIVATE|MAP_DENYWRITE , 3 , 0x0) = 0xad3e9000
mmap2(0xad599000 , 12288 , PROT_READ|PROT_WRITE , MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE , 3 , 0x1af) = 0xad599000
mmap2(0xad59c000 , 10780 , PROT_READ|PROT_WRITE , MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS , -1 , 0x0) = 0xad59c000
close(3) = 0
```

Logic(with multithreading support enabled):

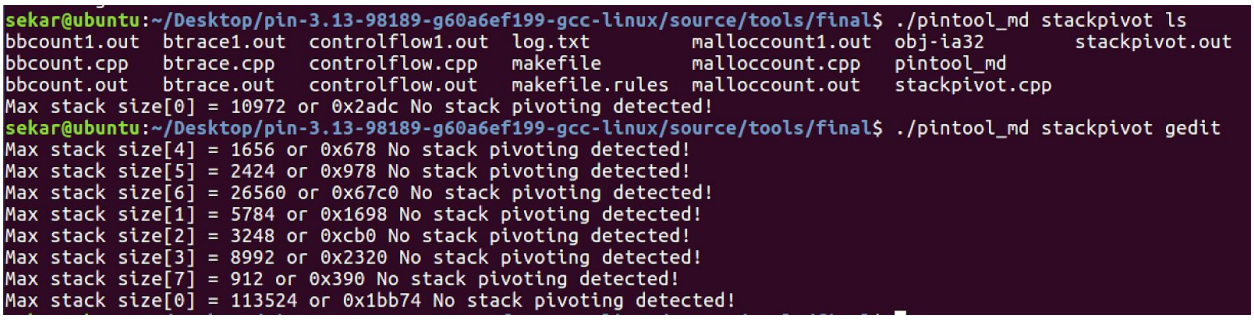
Implemented most frequent system calls like read, write, open, close, access, brk, ioctl, munmap,clone, uname, mprotect, _llseek, writev, poll, rt_sigaction, rt_sigprocmask, ugetrlimit,mmap2, stat64, fstat64, getdents64, futex, set_thread_area, exit_group, set_tid_address, statfs64, set_robust_list

I have checked the last instruction of every basic block and if it is a system call, instrument it. The entry analysis function gets the values from the registers and prints the arguments of the system call, sets the system_encountered flag as true. For return value, I instrumented every basic block beginning, and in the analysis function checked if the system_encountered for the respective thread was set. If so then print the return value else skip.

Security application 2 (stack pivoting):

../../../pin -t obj-ia32/stackpivot.so -- ls
cat stackpivot.out
../../../pin -t obj-ia32/stackpivot.so -- ./recursion 5
cat stackpivot.out

Below output for max stack size for every thread and output if stack pivoting is detected or not



Logic (with multithreading support enabled):
For checking the maximum stack usage, I have instrumented every instruction which modifies the stack pointer using INS_RegWContain api . Instrumentation is done after every such instruction to get the updated value of esp. The maximum difference is maintained which will be the maximum stack usage.

For stack pivoting, we should check on both sides of the stack.
If **min esp - current esp > some threshol**d, it might be pivoted to the heap.
Or **current esp - max esp > some threshold**, it might exceed the actual stack area used by the program to point to some malicious program. In both these cases, stack pivoting is detected.

Allowing some threshold for current esp to be greater than max esp because during dynamic loading time, the stack pointer may slightly go above the base pointer which should not be flagged as stack pivoting.