Iraqi Journal of Statistical Science (20) 2011
The Fourth Scientific Conference of the College of Computer Science & Mathematics
pp [321-335]

Detecting and Administrating Hide Processes in Linux System

Rawaa Putros Qasha*

ABSTRACT

Hiding processes in Linux system is an essential part of rootkits actions and malicious program. So, it is very important to monitor and administrate the system hidden processes to ensure the safety and reliability of the computer system. Also, process administration can be a vital factor in determining the stability of a running system.

The aim of this research is to detect hide processes in Linux system depending on /proc system files and offer tools for monitoring these processes in addition to monitoring and administrating all other processes in the system to ensure that the required processes are running and that the total number of each type of running process is appropriate to maintain system stability.

The software offers capability for displaying processes in the system including hidden processes with full information about them. And it offers options for killing or suspending a process, change process priority and viewing the memory map and the memory status for a given process.

The work has been run successfully on Linux operating system, Ubuntu distribution, version 10.4, and developed using C++ GUI programming with Qt 4 package and number of shell commands.

الخلاصة

يعد إخفاء المعالجات في نظام لينكس جزءاً أساسياً من فعاليات rootkits والبرامج الخبيثة. لذلك من المهم جدا مراقبة التحكم بالمعالجات المخفية في النظام للتأكد من سلامة و معولية نظام الحاسوب. فالتحكم بالمعالجات من الممكن أن يكون عاملا حيويا في تحديد استقرارية وثباته النظام.

الهدف من هذا البحث هو اكتشاف المعالجات المخفية في نظام لينكس بالاعتماد على ملفات النظام proc/ وتطوير برمجيات توفر أدوات لمراقبة هذه العمليات فضلا عن مراقبة والسيطرة على بقية المعالجات في النظام للتأكد من أن العمليات المطلوبة تعمل وأن العدد الإجمالي لكل نوع من العمليات يكون مناسبا للحفاظ على استقرار النظام.

* Lecturer\ College of Computers Sciences and Math.\ University of Mosul

Received:1/10 /2011 ____________________Accepted: 21 /12 / 2011

يوفر البرنامج إمكانية عرض العمليات في النظام بما في ذلك العمليات الخفية مع معلومات وافية عنها. ويقدم خيارات متعددة لإنهاء و تعليق عملية معينة وكذلك تغيير أسبقية العمليات وعرض خريطة الذاكرة وحالة الذاكرة.

تم تطبيق هذا العمل بنجاح على نظام التشغيل لينوكس ، بتوزيعة أوبونتو ذات الإصدار 10.4 ، وطور النظام باستخدام C++ برمجة واجهة المستخدم الرسومية مع حزمة QT4 وعدد من أوامر shell.

1. Introduction

Linux is a member of the large family of Unix-like operating systems, which are a multiprocessing and multi-user operating system.

Like every multiprocessing system, Linux adopts the magical effect of an apparent simultaneous execution of multiple processes [1].

As a multi-user system, Linux allows many users to access the system at the same time. Each user can run many programs, or even many instances of the same program, at the same time. The system itself runs other programs to manage system resources and control user access [14]. Therefore the system is full of processes either user or system process.

Malware needs to be executing on a system before it can perform any damage to it. An infected file cannot damage the system unless it gets executed in some or the other manner. As a result, the detection of any malware that is running on the system is of paramount importance. The most common manner in which a binary is executed is in the form of a process. Rootkits try and hide the presence of this process from the system.

A hidden process executes in a manner that is unobservable by many of the operating system's accounting and reporting functions [10].

Hidden process control, or the ability to inspect a running process and alter its execution, is a basic requirement security tool may require controlling opportunities. All such requirement necessitates the ability to observe the current state of a running process and to impose changes in its execution [17].

In order to fine-tune the system, it's important to know what is currently running, which resources are hidden, and when processes start up. From there, you can tweak configurations: disable undesirable processes, enable necessary housekeeping, and adjust your kernel to better handle your needs.

The default Ubuntu installation includes some basic processes that check devices, tune the operating system, and perform housekeeping. Some of these processes are always running, while others start up periodically, which may produce unexpected processes that can cause huge problems; administrators should know exactly what is running and when [5].

2. Process Concept:

The concept of a process is fundamental to any multiprogramming operating system [1].

A process can be defined either as "an instance of a program in execution" or as the "execution context" of a running program.

It is a collection of data structures that fully describes how far the execution of the program has progressed.

Processes are like human beings: they are generated, they have a more or less significant life, each process has just one parent and they optionally generate one or more child processes, and eventually they die [12].

Processes are object code in execution: active, alive, running programs. But they're more than just object code—processes consist of data, resources, state, and a virtualized computer [2].

2.1 Process in Linux

A process is the abstraction used by Linux to represent a running program. It's the object through which a program's use of memory, processor time, and I/O resources can be managed and monitored [3][5].

When a process is created, it is almost identical to its parent. It receives a (logical) copy of the parent's address space and executes the same code as the parent, beginning at the next instruction following the process creation system call [1].

2.1.1 Components of a Process:

A process consists of an address space and a set of data structures within the kernel. The kernel's internal data structures record various pieces of information about each process. Some of the more important information are [4][10]:

- PID: process ID number the kernel assigns a unique ID number to every process. Most commands and system calls that manipulate processes require you to specify a PID to identify the target of the operation. PIDs are assigned in order as processes are created.
- PPID: (parent PID) is the PID of the parent from which it was cloned.
- The process's address space map
- The current status of the process
- The execution priority of the process
- Information about the resources the process has used
- Information about the files and network ports that the process has opened

Some of these attributes may be shared among several processes to create a “thread group,” which is the Linux analog of a multithreaded process in traditional UNIX [5].

2.1.2 Process Scheduling

All preemptive multitasking operating systems, including Linux, implement a priority scheme for scheduling. The Linux kernel uses a process scheduler to decide which process will receive the next time slice. It does this using the process priority [neil].

Linux schedules the parent and child processes independently. And it promises that each process will run eventually—no process will be completely starved of execution resources [18].

Linux does not schedule processes willy-nilly. Instead, applications are assigned priorities that affect when their processes run. UNIX has historically called these priorities nice values, lowering a process’ priority, allowing other processes to consume more of the system’s processor time [16].

A process’s scheduling priority determines how much CPU time it receives. The kernel uses a dynamic algorithm to compute priorities, taking into account the amount of CPU time that a process has recently consumed and the length of time it has been waiting to run [5]

2.1.2.1 Niceness

The “niceness” of a process is a numeric hint to the kernel about how the process should be treated in relationship to other processes contending for the CPU. The nice value is a property that exists for every process. It is not, as is often misunderstood, the priority of a process. It is a number that influences the priority property of a process [5].

The nice value dictates when a process runs. Linux schedules runnable processes in order of highest to lowest priority: a process with a higher priority runs before a process with a lower priority. The nice value also dictates the size of a process’ time slice. Legal nice values range from –20 to 19 inclusive, with a default value of 0. The lower a process’ nice value’, the higher its priority.

You may specify that a process is less important—and should be given a lower priority by assigning it a higher niceness value. A higher niceness value means that the process is given a lesser execution priority; conversely, a process with a lower (that is, negative) niceness gets more execution time [11].

3. The /proc File System:

Linux systems support a special file system called /proc. /proc is a window into the running Linux kernel. Files in the /proc file system don't correspond to actual files on a physical device. Instead, they are magic objects that behave like files but provide access to parameters, data structures, and statistics in the kernel [11][12].

The "contents" of these files are not always fixed blocks of data, as ordinary file contents are. Instead, they are generated on the fly by the Linux kernel when you read from the file [2].

The /proc file system holds information on each running process as well as hardware-related information on your system.

Each process has a subdirectory under /proc named after its process ID, which exists for the lifetime of the process. Images of active processes are stored here by their process ID number contains detailed and technical information [9].

The /proc file system started out holding just process information. Now it holds all sorts of operating system and hardware data.

The /proc file system appears to be a normal directory on disk. Inside /proc each process, for example, has a directory under /proc. The directory name is the process ID number [3][17].

Process-specific information is divided into subdirectories named by PID. These directories appear and disappear dynamically as processes start and terminate on the system. Each directory contains several entries providing access to information about the running process. From these process directories the /proc file system gets its name [5][neil]. Some of these subdirectories are used in this work, as follows:

1. Stat : General process status information, contains details information about the process itself, and a lot of status and statistical information about the process. Table (1) show most usable information in stat subdirectory.

Table (1) some fields of stat subdirectory

Field	Content
Pid	Process id
Tcomm	filename of the executable
State	state (R is running, S is sleeping, D is sleeping in an uninterruptible wait, Z is zombie, T is traced or stopped)
Ppid	process id of the parent process
Nice	nice level
start_code	address above which program text can run

2. Statm : Memory usage information, contains more detailed information about the process memory usage. Its seven fields are explained in table (2).

Table (2) Contents of the statm

Field	Content
size	total program size
resident	size of memory portions (pages)
shared	number of pages that are shared
trs	number of pages that are 'code'
lrs	number of pages of library
drs	number of pages of data/stack
dt	number of dirty pages

3. pmap: Memory mapping information, displays information about files mapped into the process's address. For each mapped file, maps displays the range of addresses in the process's address space into which the file is mapped, the permissions on these addresses, the name of the file, and other information as shown in the following format[11]:

```
address            perms  offset    dev    inode    pathname

08048000-08049000  r-xp   00000000  03:00  8312     /opt/test
08049000-0804a000  rw-p   00001000  03:00  8312     /opt/test
 .
 .
```

Where:
"address": is the address space in the process that it occupies.
"perms": is a set of permissions, r = read, w = write, x = execute
s = shared, p = private (copy on write).
"offset": is the offset into the mapping.
"dev": is the device (major:minor), and
"inode": is the inode on that device. 0 indicates that no inode is associated with the memory region, as the case would be with BSS (uninitialized data).
"pathname": shows the name associated file for this mapping.

4. Signals in Linux

Signals are mechanisms for communicating with and manipulating processes in Linux. A signal is a special message sent to a process. Signals are asynchronous; when a process receives a signal, it processes the signal immediately, without finishing the current function or even the current line of code. There are several dozen different signals, each with a different meaning. Each signal type is specified by its signal number, but in programs, signal can be referred to by its name. In Linux, these are defined in /usr/include/bits/signum.h. A process may also send a signal to another process [9][11].

Signals are a mechanism for one-way asynchronous notifications. A signal may be sent from the kernel to a process, from a process to another process, or from a process to itself. Signals typically alert a process to some event, such as a segmentation fault, or the user pressing of Ctrl-C [16].

The Linux kernel implements about 30 signals (the exact number is architecture dependent), table (3) lists a number of Linux signals. Each signal is represented by a numeric constant and a textual name [6][neil].

Table (3) Some of Linux Signals

Signal	**Name Description**
SIGALRM	Alarm clock
SIGHUP	Hangup
SIGINT	Terminal interrupt
SIGABORT	*Process abort
SIGKILL	Kill (can't be caught or ignored)
SIGQUIT	Terminal quit
SIGTERM	Termination

Signals are process-level interrupt requests. About thirty different kinds are defined, and they're used in a variety of ways [5]:

• They can be sent among processes as a means of communication.
• They can be sent by the terminal driver to kill, interrupt, or suspend processes
• They can be sent by the administrator (with kill) to achieve various results.
• They can be sent by the kernel when a process commits an infraction such
as division by zero.
• They can be sent by the kernel to notify a process of an "interesting" condition such as the death of a child process or the availability of data on an
I/O channel.

6. Implementing Hidden Process Administration

Process management in Linux was achieved using system call functions and dealing with /proc system files or bash shell commands. The software is developed using Qt package, which is a cross-platform, graphical, application development toolkit that enables compiling and running applications on Windows, Mac OS X, Linux, and different brands of Unix[15][10].

A large part of Qt is devoted to provid a platform-neutral interface to everything, ranging from representing characters in memory to creating a multithreaded graphical application [8].

Process administration software includes the following parts:

- Detecting Hidden Process: all process in the system and information about each one is extracted from /proc through /stat folder for each process including: PID (process ID), stat), PPID process parent ID, Nice value, either the process is hidden or not, and executable path.

Extracting information from /proc is performed as follows:

```
QDir dir("/proc");
    QFileInfoList list = dir.entryInfoList();
    for(int i=1; i<list.size();++i)
    {
       QFileInfo fileInfo = list.at(i);
       if(Hidden == 1)
          get_OnlyHide(fileInfo.fileName());
       else
          get_task(fileInfo.fileName());
    }
```

Linux does not provide any tools or commands to display hidden processes in the system. So to display hidden process in this work, the following steps are performed:

1- Running ps shell command to obtain a snapshot of the process in the system except hidden ones [13][15].

2- Obtaining all processes in the system including hidden processes from /proc file system.

3- Compare between the results from step 1 and 2, then display every process produced only from step 2 and not contained from the result of ps command, then display hidden process information.

```
    void ProcessAdmin::ShowHidden()
    {   // run ps command to compare result with /proc
         p->start("ps -e");
    }
    void ProcessAdmin::get_OnlyHide(QString id)
    {
       QString fileName ="/proc/"+id+"/stat";
       int f=0;
       QFile file(fileName);
       if(file.open(QIODevice::ReadOnly))
       {
        QTextStream stream(&file);
        QString line;
        line = stream.readLine();
        QStringList list =  line.split(" ");
        foreach(QString pid, pidList)
           if(pid == list[0])
              f=1;
        if(f==0)
        {
           treeNode = new QTreeWidgetItem(ui->treeWidget);
           treeNode->setText(0, list[0]); // PID
           treeNode->setText(1, list[1]); // name
           treeNode->setText(2, list[2]); // stat
           treeNode->setText(3, list[3]); // PPID
           /// get priority
           bool ok;
           int ret = getpriority(PRIO_PROCESS,list[0].toInt(&ok,10));
           treeNode->setText(4, QString::number(ret));
           treeNode->setText(5, "True");
        }
        file.close();
     }
  }
```

- Starting new process: depending on a process name supported by the user:

```
QProcess run;
run.execute(ui->lineEdit->text());
```

- Monitoring process execution and termination by offering the following options:
  - Kill a single process either using process ID or name [13]. Killing operation is performed by sending a signal to the process using kill system call function as follows:

```
void ProcessAdmin::KillProc()
{
   int pid = ui->treeWidget->currentItem()->text(0).toInt();
   kill(pid,SIGTERM);
}
```

Where SIGTERM: is the signal to be send to the process to terminate it:

- Kill a process and all its descendent children, a recursive function is developed to search for all children for a given process, then kill the child starting from the last one.

```
void ProcessAdmin::recuKill(int pid)
{
   QDir dir("/proc");
   int ppid,childId;
    QFileInfoList list = dir.entryInfoList();
    for(int i=0; i<list.size();++i)
    {
       QFileInfo fileInfo = list.at(i);
       QString fileName ="/proc/"+fileInfo.fileName()+"/stat";
       QFile file(fileName);
       if(file.open(QIODevice::ReadOnly))
       {
          QTextStream stream(&file);
          QString line;
          line = stream.readLine();
          QStringList list =  line.split(" ");
          QString ppids = list[3];
          ppid = ppids.toInt();
          if(pid == ppid)
          {
             QString child = list[0];
             childId = child.toInt();
             recuKill(childId);
             kill(childId,SIGTERM);
          }
        }
    }
    kill(pid,SIGTERM);
}
```

- suspend and resume a process:

```
kill(pid,SIGSTOP);     //To suspend a process
kill(pid,SIGCONT);     //To resume a suspended process
```

➢ Scheduling process by changing its priority through process nice value using setpriority system function:

```
void ProcessAdmin::SetPriority()
{
   int nice = ui->lineEdit_2->text().toInt();
```

```
    int pid = ui->treeWidget->currentItem()->text(0).toInt();
    int ret;
    ret = setpriority(PRIO_PROCESS,pid,nice);
    if(ret == -1)
        QMessageBox::information(this ,"error","error ");
}
```

- Displaying memory map for any process by exploring the information stored by the system in folder pmap in the /proc system folder as follows:

```
void pmapinfo::get_task(QString id)
{
    QString fileName ="/proc/"+id+"/maps";
    int f=0;
    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly))
    {
        QTextStream stream(&file);
        QString line;
        line = stream.readAll();
        ui->listWidget->addItem(line);
        file.close();
    }
}
```

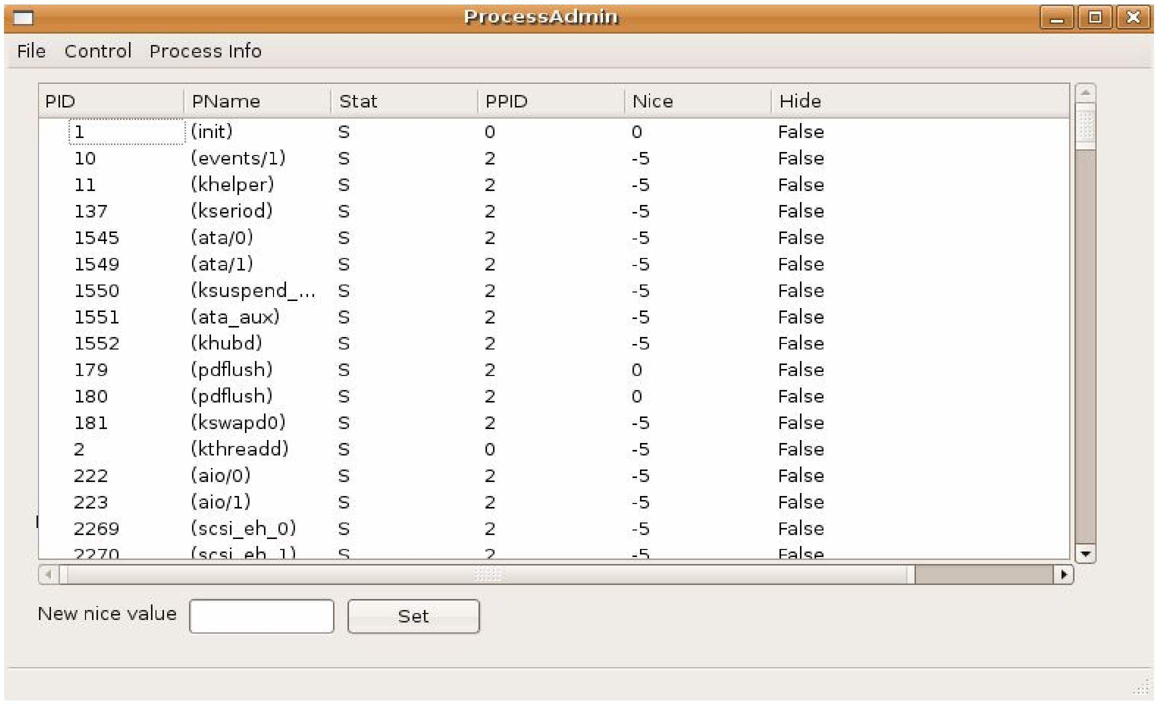
- Displaying Process memory status information that is located in Statm folder in /proc file system for any selected process:

```
void ProcessMem::get_task(QString id)
{
    QString fileName ="/proc/"+id+"/statm";
    int f=0;
    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly))
    {
        QTextStream stream(&file);
        QString line;
        line = stream.readLine();
        QStringList list =  line.split(" ");
        treeNode = new QTreeWidgetItem(ui->treeWidget);
        treeNode->setText(0, id);
        treeNode->setText(1, list[1]);
        treeNode->setText(2, list[2]);
        treeNode->setText(3, list[3]);
        treeNode->setText(4, list[4]);
        treeNode->setText(5, list[5]);
        treeNode->setText(6, list[6]);
        treeNode->setText(7, list[7]);
        file.close();
      }
    }
```

6. Hidden Process Administration Software:

Research application was developed using QT package. It contains the following windows:

1. Main application windows as shown in figure(1), contains the following menu options:
   - File: includes:
     - Show submenu includes:
       - All: to view full information about each.
       - Hidden: to display only hidden processes.
     - New task: To start a new process.
     - Exit
   - Control : includes:
     - Kill: to kill a single process.
     - Kill ptree: to kill process and all its descending children.
     - Suspend: to suspend a process.
     - Resume: to resume a suspended process.
   - Process info: includes:
     - Process memory map: to view process memory map info.
     - Process memory status: to view process memory status info.

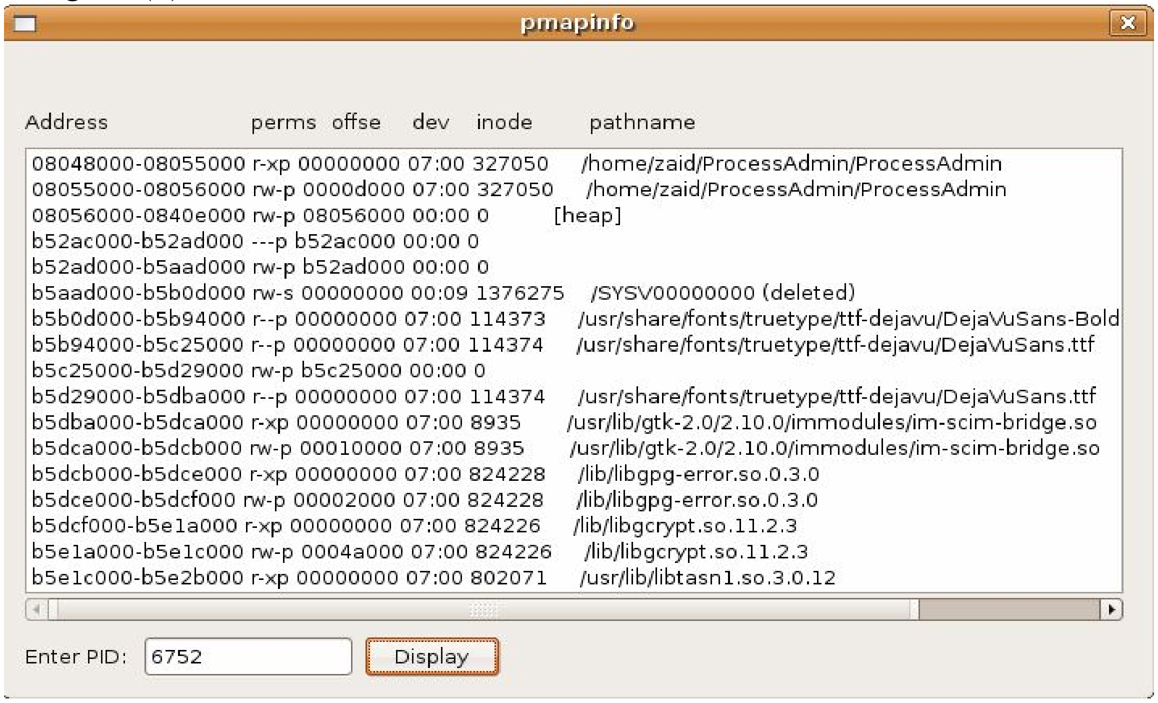


PID	PName	Stat	PPID	Nice	Hide
1	(init)	S	0	0	False
10	(events/1)	S	2	-5	False
11	(khelper)	S	2	-5	False
137	(kseriod)	S	2	-5	False
1545	(ata/0)	S	2	-5	False
1549	(ata/1)	S	2	-5	False
1550	(ksuspend_...	S	2	-5	False
1551	(ata_aux)	S	2	-5	False
1552	(khubd)	S	2	-5	False
179	(pdflush)	S	2	0	False
180	(pdflush)	S	2	0	False
181	(kswapd0)	S	2	-5	False
2	(kthreadd)	S	0	-5	False
222	(aio/0)	S	2	-5	False
223	(aio/1)	S	2	-5	False
2269	(scsi_eh_0)	S	2	-5	False
2270	(scsi_eh_1)	S	2	-5	False

Figure (1) Process Administration Software main window

To change the nice value for a selected process, a new nice value must be entered through the text box then the set button must be pressed.

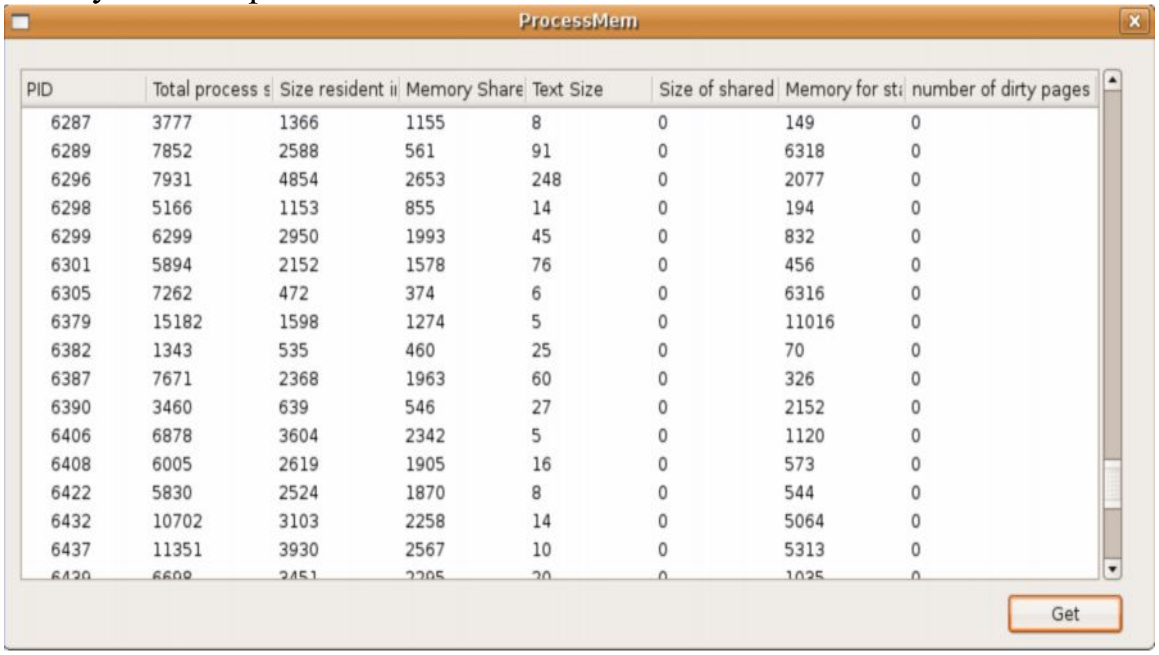
2. The second window is used to display selected process memory map as shown in figure (2).



```
Address                  perms  offse    dev    inode       pathname
08048000-08055000 r-xp 00000000 07:00 327050     /home/zaid/ProcessAdmin/ProcessAdmin
08055000-08056000 rw-p 0000d000 07:00 327050     /home/zaid/ProcessAdmin/ProcessAdmin
08056000-0840e000 rw-p 08056000 00:00 0          [heap]
b52ac000-b52ad000 ---p b52ac000 00:00 0
b52ad000-b5aad000 rw-p b52ad000 00:00 0
b5aad000-b5b0d000 rw-s 00000000 00:09 1376275    /SYSV00000000 (deleted)
b5b0d000-b5b94000 r--p 00000000 07:00 114373     /usr/share/fonts/truetype/ttf-dejavu/DejaVuSans-Bold
b5b94000-b5c25000 r--p 00000000 07:00 114374     /usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf
b5c25000-b5d29000 rw-p b5c25000 00:00 0
b5d29000-b5dba000 r--p 00000000 07:00 114374     /usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf
b5dba000-b5dca000 r-xp 00000000 07:00 8935      /usr/lib/gtk-2.0/2.10.0/immodules/im-scim-bridge.so
b5dca000-b5dcb000 rw-p 00010000 07:00 8935      /usr/lib/gtk-2.0/2.10.0/immodules/im-scim-bridge.so
b5dcb000-b5dce000 r-xp 00000000 07:00 824228     /lib/libgpg-error.so.0.3.0
b5dce000-b5dcf000 rw-p 00002000 07:00 824228     /lib/libgpg-error.so.0.3.0
b5dcf000-b5e1a000 r-xp 00000000 07:00 824226     /lib/libgcrypt.so.11.2.3
b5e1a000-b5e1c000 rw-p 0004a000 07:00 824226     /lib/libgcrypt.so.11.2.3
b5e1c000-b5e2b000 r-xp 00000000 07:00 802071     /usr/lib/libtasn1.so.3.0.12
```

Figure (2) Process Memory Map window

3. Third window depicted in figure (3), is used to view memory status information for any selected process.



PID	Total process s	Size resident i	Memory Share	Text Size	Size of shared	Memory for st	number of dirty pages
6287	3777	1366	1155	8	0	149	0
6289	7852	2588	561	91	0	6318	0
6296	7931	4854	2653	248	0	2077	0
6298	5166	1153	855	14	0	194	0
6299	6299	2950	1993	45	0	832	0
6301	5894	2152	1578	76	0	456	0
6305	7262	472	374	6	0	6316	0
6379	15182	1598	1274	5	0	11016	0
6382	1343	535	460	25	0	70	0
6387	7671	2368	1963	60	0	326	0
6390	3460	639	546	27	0	2152	0
6406	6878	3604	2342	5	0	1120	0
6408	6005	2619	1905	16	0	573	0
6422	5830	2524	1870	8	0	544	0
6432	10702	3103	2258	14	0	5064	0
6437	11351	3930	2567	10	0	5313	0

Figure (3) Process Memory Status window

7. Related Works:

Many efforts and works had been made to detect hidden processes in different systems. James, Jeffrey and John present a method for detecting hidden processes, whether running on the Windows or Linux operating system, requires the examination of each thread to ensure that its corresponding process descriptor is appropriately linked. Accordingly, both require added functionality to the operating system [7]. In [19] the researchers are presented a new VMbased approach called Libra to detect hidden processes implicitly. Like previous VM-based security services, Libra is resilient to kernel-mode guest malware attack by virtue of its location within a VMM layer.

8. Conclusion

This research supports interface that deals with a /proc file on Linux. This interface allows tracing processes to read detailed process state from. Our hidden process control implementation supports both a larger set of operations and more finely-grained control than that are offered by ps terminal command. Extracting process information from /proc programmatically is difficult to implement also running shell command from the QT and obtain its results is a hard task.

Hidden Process Detection and administration tools for Linux system can be considered an important part in the system to ensure that the required processes are running and any unwanted processes can be controlled.

References

1. Daniel P. Bovet, 2005, "Understanding the Linux Kernel", Marco Cesati, O'Reilly.
2. Dee-Ann LeBlanc, 2002, "Linux System Administration Tools", http://www.linuxjournal.com/article/5918.
3. Eric Foster-Johnson, John C. Welch, and Micah Anderson, 2005, "Beginning Shell Programming", Wiley Publishing, Inc.
4. Eric Uday Kumar, "User-mode memory scanning on 32-bit & 64-bit windows", 2010, J Comput Virol (2010) 6:123–141 DOI 10.1007/s11416-008-0091-3.
5. Evi Nemeth, Garth Snyder, and Trent R. Hein, 2007, "Linux Administration handbook", Pearson Education, Inc, 2nd Ed.
6. Graham Glass and King Ables, 2006, "Linux for Programmers and Users", Prentice Hall.
7. James Butler; Jeffrey L. and John Pinkston, "HIDDEN PROCESSES: The Implication for Intrusion Detection", 2003, 2003 IEEE ISBN 0-7803-7808-3.
8. Jasmin Blanchette and Mark Summerfield, 2008, "C++ GUI Programming with Qt 4", Prentice Hall, 2nd Edition.
9. John Fusco, 2007, "The Linux Programming Toolbox", Pearson Education, Inc.
10. Linux Reference Manual, Section 5, proc.
11. Mark Mitchell, Jeffrey Oldham, and Alex Samuel, 2001, "Advanced Linux Programming", New Riders Publishing.
12. Matthew West, 2005, "System administration", The Shuttleworth Foundation.
13. Neal Krawetz, 2007, "Hacking Ubuntu", Wiley Publishing, Inc.
14. Neil Matthew & Richard Stones, 2007, "Beginning Linux Programming", Wiley Publishing Inc, 4th Edition.
15. Philip J. Hollenback, 2008, "Process monitoring with ps-watcher", http://www.linux.com/archive/feature/148189.
16. Robert Love, 2007, "Linux System Programming", O'Reilly Media Inc.
17. Ron Peters, 2009, "Experts Shell Script", APress.
18. Simone Demblonand and Sebastian Spitzner, 2004, "Linux Internals", the Shuttleworth Foundation.
19. Wei Li, Long Chen; Hongjiang Ji; Tong Zhang, "Improvement of Real-Time Process Monitor Technology on Linux Based on Mandatory Running Control", 2011, IEEE 978-1-4244-8728-8/1.
20. Yan W.; Jinjing Z.; Huaimin W., "Implicit Detection of Hidden Processes with a Local-Booted Virtual Machine", 2008, International Conference on Information Security and Assurance, IEEE DOI 10.1109/ISA.2008.22.