Development of an Enhanced Security Strategy for Linux Hosts

Dmitri Podkorytov,
Computer Science Department
Kurgan State University, Russia

Dennis Guster,
BCIS Department
St. Cloud State University, MN, USA

Paul Safonov,
BCIS Department
St. Cloud State University, MN, USA

Alina Rudenko,
Network Engineering
St. Cloud State University, MN, USA

Abstract

An enhanced strategy for improving the security on a Linux host is described. It is founded upon expanding the projection logic beyond setting single attributes within the user, group or resource level. A "sandpit" logic is developed that is designed to minimize the effect of resource intensive attacks such as denial of service. A number of coding examples are provided to illustrate the basic structure of the system.

Introduction

Although successive releases of the Linux operating system have been responsive to holes exploited by hackers there are still many vulnerabilities in the system. This may be attributed in part to the security design of the most modern operating systems in which values within the following categories are set:

- User
- Groups of Users
- Resources
- Access rights of Groups or Users within the Resource.

This scheme of granting rights is commonly used and in many cases is sufficient. However, since this kind of scheme has been adopted, a number of different vulnerability scenarios have been detected. In these cases, rights schemes have no way of protecting the system. These attack scenarios are:

- Denial of Service attacks
- Viruses
- Attacks that cause stack failures and gain access to a running (working) process

Therefore, based on the prevalence of these attacks it is clear that there is a need for a system capable of a more sophisticated security plan. To meet this need a system is proposed herein based on the principle of a task manager that monitors and modifies the behavior of system processes.

Often process protection is based solely on the owner attribute which is just one of more then 100 process attributes. We propose a scheme that will allows us to build an ordinal scheme of several of these attributes in the User – Rights – Resources categories as well as more complex schemes that are based on periodic checks of the status of a process to ascertain if it is in a dangerous state. The goal of the system would not just be identification of dangerous states, rather upon identification it should have available a mechanism to automatically pass the process to a safe state.

Methodology

In an attempt to provide such functionality the current project was undertaken to provide automatic control of processes. The basis of the project is centered on the fact that the system has a primary (common) description of valid parameters of a given process and in case these parameters are out of the acceptable limits, a signal is dispatched to end or suspend such process(es). Thus the initial purpose of the project is the creation of a system that will limit behavior of the process in the system using a set of rules. This is a further development of widely used technique: starting a process in a "sandpit" where activity of that process can not exceed the limits defined by the control system. When using this "sandpit" logic the following examples apply.

1. Regarding the behavior of a process: these rules are applied disregarding the activities of other system processes. In the default operating system: “a process can derivate descendants”, however when using the “sandpit” logic “the process can derivate descendants if the total number of processes in the system is no more then n”.
2. Rules are static, without taking into account the dynamic behavior of the process itself. For example: in the default system “it is possible to go through this port”, but in the “sandpit” it may only be possible to go through the port no more than two times per one second”.

When implementing this approach it is necessary to follow the following process guidelines.

1. They must possess relational integrity in metadata representation (data about data) and describe critical factors about your systems and applications, such as where a particular data source is located and the data types that are used by these systems and applications) [1].

2. Functional dependence among processes may occur depending on their behavior, prior history and current state (To determine functional dependency, you can think of it like this: Given a value for Field A, can you determine the single value for B? If B relies on A, then A is said to functionally determine B) [2].

Complete implementation of these principles is a difficult task, because in a traditional Von Neumann's architecture the time factor is always a limitation and a solution for any task must take place within the allocated time frame. In processing results from databases it is still theoretically possible to achieve relational entirety because the data table has a finite sort. However, in our case, to achieve a complete description of the system we need to operate with an infinite table, because for each event it is necessary to have a time mark and the attributes for each process and the system state recorded. Therefore, the task manager has to maintain all states of the system dynamically in memory and do it efficiently if performance is to remain adequate.

There are three theoretical concepts that will be useful in the project development:

1. Recursive calculations - calculations allow the system to (it can be rolled back by any amount of steps (n)) determine what state was in a previous instant of time.
2. The theory of quantum calculations.
3. Searches using architectures with natural parallelism of calculations, when any transition of the system takes place from one state to another the calculation can be done for one single step or in parallel for a varying quantity of steps.

The task of recognition of states of the system and transformation from a dangerous to safe state is a derivative of the task of language recognition which is traditionally considered as a NP-complete problem [3]. Therefore modern methods of solving NP-complete problems also are related to determining the solution for our task. Such, works are listed in the references as [5-15].

A numeration of functions, as offered by the primary author in publications [4,5], is used to describe the bounds of behavior of the system. The description of an information system based on the numeration of functions enables us to more effectively (in comparison with the theory of sets) handle incoming information. The cost of this increased efficiency is more real memory. In other words, there is a tradeoff: additional memory for reduced processing time. Employing this methodology allows us to achieve a system approaching the efficiency of quantum calculators.

Application of the Task Management System

Specifically, the system should be able to limit process behavior such as in the following examples.

1. prohibit execution of demons such as ftpd or sshd depending on the time of day, load on the system or activity through the network interfaces;
2. prohibit processes that have administration rights and are bound to

the terminal ptty* for 12 hours;
3. “close” and “open” network interfaces by the defined rules;
4. know how not only to stop processes, but also how to temporarily suspend them and also to restart their execution; and
5. slow down execution of any process which is exceeding the rules of time allocation (likely a DoS attack).

The development strategy of the system proceeded with the understanding that the system needed be adaptable to distributed computing architectures using natural parallelism. Therefore it will be suitable for the construction of an enterprise level security network, consisting of a considerable number of handled objects.

The system is modular. It allows development of new functions and consequently will scale well, and be functionally extendable/transferable. Such flexibility enables usage of the system to accomplish complex security tasks involving multiple attributes, a vast improvement over single attribute strategies. A number of examples follow that are designed to illustrate the structure of the system.

The most trivial example of the enhanced system is:

psdump | filter[0] .. filter[N] | pskill OPTIONS

where:

psdump - printing of the attributes of all processes in the system.

filter [i] - stream filters such as grep, awk, etc. These filters specify the rules of passing the stream for those processes which do not meet the 'safe' configuration.

pskill – sends kill signals to processes. The descriptor of the process is read from stdin, after that a signal for closing is sent to this process.

In general, the system structure consists of:

1. Sensors - programs for collection of information about the system. Basically, inputs for feedback from the process control system. For example: psdump.
2. Transport units. As illustrated in Figure 1, they are a function of the vertical and horizontal buses. Horizontal buses have one data source (Sensor) and a number of consumers (Vertical buses). More specifically, it is a bus with a common input and in the UNIX file system can be created as a fixed (named) channel.

For example:

#Sensor1 | tee pipe[1] pipe[2] ... pipe[N]

The Vertical bus has several inputs and one output (Actor) which is a bus with a single output destination.

#eet pipe[1] ... pipe[N] | Actor

Where eet is a “shell” script:

```
for $i from 1 to N
  do
  (cat pipe[$i] &)
  done
```

The matrix of filters (tm-grep, grep etc.) where the array is defined: filter [i, j] prohibits or allows the transfer of data from the horizontal bus i to the vertical bus j.

3. The common example of this matrix is a list with one sensor and one actor.
4. Actors - these units are created for system control. They evaluate the information and provide management of the system. The most commonly used actors are: pskill, psfreeze, which potentially allow a misbehaving process to be killed or suspended.
5. Counters are units with memory. Activity on the counter depends on its prehistory (for example data coming from stream stdin). In general it is a n-bit counter that is joined with some logical function that describes the process state. For most logical functions we can interpret the behavior of the counter in a timely manner. Without counters the system could be classified as just simple logical automaton.

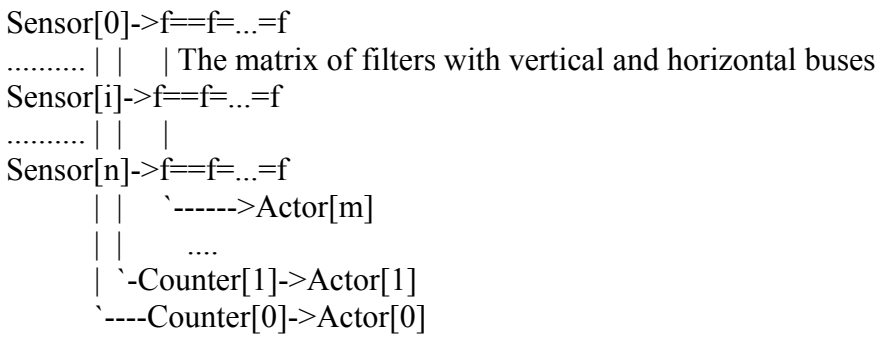

Figure 1. Bus Structure

Operation on Multiple Nodes

The grid consists of:
1) Vertical buses
2) Horizontal buses.

The Horizontal bus is a channel with one input and multiple outputs. It looks like the tee program in Unix systems. Our implementation of this bus is named tcp-tee.
This module passes stdin to TCP addresses and their related ports.

The basic example of transmitting process activity to others systems:
psdump | tcp-tee 10.1.1.1:21 10.1.1.2:21 10.1.1.3:21

The network module is a vertical bus. It collects information from multiple nodes and sends it to one node which prints it to stdout.

The Module is named tcp-eet and an example follows:

```
tcp-eet 10.1.1.1:21 10.1.1.2:21 10.1.1.3:21 | pskill
```

This example collects information from port 21(ftp) on nodes from 10.1.1.1 to 10.1.1.3 and then passes the data to actor pskill.

The logic could also be used to build a computing grid (on the process level) on a single node using the loopback:

```
psdump | tcp-tee 127.0.0.1:1001 127.0.0.1:1002 127.0.0.1:1003 &
sockdump | tcp-tee 127.0.0.1:2001 127.0.0.1:2002 127.0.0.1:2003 &

tcp-eet 127.0.0.1:1001 127.0.0.1:2001 | var-greep sensor EQ psdump | var-greep
tty LIKE *pty* |var-greep uid NE 0 |tm-greep hour GR 20 | pskill &
tcp-eet 127.0.0.1:1002 127.0.0.1:2002 | var-greep sensor EQ psdump | var-greep
cpu GR 80.0 | var-greep tty LIKE *tty* | psfreeze 10 100 &
tcp-eet 127.0.0.1:1003 127.0.0.1:2003 | var-greep sensor EQ netdump |var-greep
port GR 800 | if_has_input "ifconfig -d ed0 " &
```

This grid structure employs a sensor to dump kernel information about opened sockets. Every sensor linked to a given port writes a stream of attribute values. Therefore we can determine the performance or vulnerability of a given stream by the various sensor values.

In this modular design arguments are passed through the various filters and the resulting command set is executed. In this example, the last line executes the "ifconfig -d ed0 " command which downs network interface ed0. The coding structure for working in a multi-node cluster looks similar to this example, but the IP addresses of the local loop interface must be replaced with addresses for external nodes.

The model described herein uses calculations that lend themselves to natural parallelism and could be easily adapted to a distributed system architecture. Sensors and Actors are input and output points in a chain that handles feedback from the cybernetic system. Their quantity can vary to a wide degree and depends on the complexity of the tasks to be solved.

The Basic Modules to Support Implementation of the System

The security system includes three major implementation modules: *Sensors, Actors*, and *Filters*.

Sensors:

psdump – prints full information about each process in the system.

This module's task is to fetch information about current processes in the system and print it to stdout. The stream is in plain text format. For each process there is a separate line.

The output might look like the following:

field0=value0 [field1=value1 [field2=value ... [fieldn=valuen] ...]]

Where the "field" is a process attribute (for example: command_line, pid, uid, ruid, gid, rgid , tty, etc.), and the "value" is the attributes value (number , word or a quoted line) .

Example:

pid=0 command="init" uid=0 gid=0 tyy=null ....
pid=235 command="sh batchfile" uid=100 gid=100 tyy=dev/pty0 ....

Actors:

The pskill command is an actor module. Its conceptual functionality is very simple.
It searches the stdin stream input line for pid=X and then sends a signal SIGTERM to end process X. It is designed to terminate the processes that exceed the defined limits.

pskill - reads the descriptor of the process from stdin and sends the specified signal.

Usage:

#pskill ACTION

ACTION:
TERM
STOP
CONT
Signal number
else run system(argv[1] ... argv[argc])

psdump|... filters ...| pskill STOP – stop the processes
psdump|... ...| pskill TERM – finish the processes
psdump|... ...| pskill mail -s alarm

cool_admin@protected.site.com – send an alert by e-mail
psdump|... ...| pskill ifconfig -d ed0 # - to close the network interface in a critical situation

psfreeze - downturn of activity of process

Usage:

...|psfreeze Activity_time Sleep_time (in sec or nsec)

Filters:

Filters are necessary to enforce the allowable limits for each process.
They also must comply with the structure of Unix stdin/stdout streams. The arguments for the filters are specified on the command line. If the stream satisfies the conditions specified in the arguments it is passed from stdin to stdout

The minimal set of filters necessary for the proposed system is just three:

The **tm-greep** filter permits or denies the flow of a stream. This filter's activity is often dependent on some temporal value.
The rules specifying these arguments are shown in the lines below.

Example:

psdump | tm-greep hour GR 21|...
- passes the stream, if system time is greater then 9 pm.

psdump | tm-greep wday LT 3|...
- passes the stream, if day of week is less than 3.

psdump | tm-greep hour GR 3| tm-greep min LT 21
- passes the stream, if system time is between 3:00 pm and 3:21 pm.

Usage:

$ process_A | tm-grep TIME_RULES | process_B

If TIME_RULES is true then stdout from process_A is passed to stdin on process_B.

So therefore this filter, can be used to restrict process activity for a specific time frame.

The **var-greep** is a filter that controls stream passing if at least one value from the process attributes satisfies the argument rules.

Example:

psdump | var-greep uid GR 0 | pskill
- kills all users processes, where the process owner is not the root.

psdump | var-greep command LIKE *ftpd* |tm-greep hour GR 20 | pskill
- denies activity of the ftpd daemon from 9:00 pm to 00:00

This command can be used to control the time any daemon is accessible, that is launched by inetd daemon.

psdump | var-greep tty LIKE *pty* |var-greep uid NE 0 |tm-greep hour GR 20 | pskill

This example denies activity of processes with non roots uid's bound to SSH using network connected terminal devices from 9:00 pm to 00:00. Therefore, this command could be used to impose time restrictions on user processes.

psdump | var-greep uid NE rid |tm-greep hour GR 20 | pskill

It denies activity for processes with "real user identifiers" different from "user identifiers" (real uid =/= uid) from 9:00 pm to 00:00. This would be an effective means of imposing time restrictions for controlling su and sudo commands.

psdump | var-greep cpu GR 80.0 |var-greep tty LIKE *tty* | pskill

In this command string, those processes using more than 80% of the CPU and that have a terminal link for output will be killed.

psfreeze - delay of process activity

Usage:

psfreeze Activity_time Sleep_time (in sec or nsec)

Example:

psdump | var-greep cpu GR 80.0 |var-greep tty LIKE *tty* | psfreeze 10 100

Allows those processes using more than 80% of the CPU and that have a terminal link for output to be delayed 10 seconds.

Conclusions

Given the degree of hostility of the internet it is important that any host that has internet connectivity be well protected. The methodology described herein provides a promising structure for that protection. Although it is designed to handle a wide variety of attacks its ability to stop or slow down denial of service attacks is especially pertinent given the frequency of these attacks and the low level of sophistication required to launch such attacks.

References

1. Morgenthal, J. & Walmsley, P. (2000). Mining for Metadata: Industry Trend or Event**.** *Software Magazine*, February. http://www.findarticles.com/cf_0/m0SMG/1_20/61298805/p1/article.jhtml

2. Lotito, J. (2001). Concepts of Database Design and Management, April. http://www.sitepoint.com/article/378

3. Manber, U. (1989). Introduction to Algorithms: A Creative Approach. *Chapter 11, NP-Completeness*, Addison-Wesley Publishing Company, 1989.

4. Podkorytov, D. (2002). Numeration System for Computing Logical Functions, in Telecommunications, *Mathematics and Information Sciences: Research and Innovations, Vol. 6, State University of Leningrad Region named after A.S. Pushkin, St. Petersburg*. pp. 125-129 ISBN 5-8290-0349-X. (in Russian).

5. Podkorytov, D. (2003). Calculation of all Permutations in an M-length Word at a Polynomial Time. Collection of post-graduate papers (in Sciences, Engineering and Economics), Kurgan State University - Kurgan: Kurgan State University. pp. 6-9. (in Russian).

6. Telpiz, M. (2001). Positional Principle for Calculus of Functions. *Vol. 1, Institute of Space Research, Russian Science Association,* Moscow. 457 p. (in Russian).

7. Lachinov, V. & Polyakov, A. (1999). Information dynamics or a way to Open Systems to the World. The St. Petersburg State Technical University. (in Russian). http://www.polyakov.com/informodynamics/index.html

8. Lachinov, V. & Polyakov, A. (1998). Information Theories: Selected Lectures on Information Dynamics Basics, St. Petersburg State Technical University. p.146. (in Russian).

9. Polyakov, A. (2001). From Quantitative Information to Information Dynamic Machines, St. Petersburg State Technical University. P. 80. (in Russian).

10. Razborov, A. (1991). Lower bounds for deterministic and nondeterministic branching programs. *Proc. FCT'91, Lecture Notes in Computer Science, 521*. Springer, Berlin. pp. 47-60.

11. Jukna, S., Razborov, A., Savicky, P., & Wegner, I. (1999). On P versus NP or co-NP for Decision Tree and Read-Once Branching Programs. *Computational Complexity*, Birkhauser Verlag, Basel.

12. Shor, P. (1994). Algorithms for Quantum Computation: Discrete Logarithms and Factoring. Proceedings, 35th, Annual Symposium on Foundations of Computer Science. Santa Fe, NM, November 20-22. IEEE Computer Society Press, pp. 124-134.

13. Shor, P. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Computer*. 26:5. pp. 1484-1509.

14. Bennet, C. (1979). Logical reversibility of computation, *IBM J. Res. Dev*. 6. pp. 525-532.

15. Deutsch, D. (1986). Three connections between Everett's interpretation and experiment. in Penrose, R. and Isham, C., (eds.), *Quantum Concepts in Space and Time*, Clarendon Press, Oxford. pp. 215-225.