Assignment #4 - Process Hunter

CSE 433/533 – Computer and Network Security
Winter 2013 – Prof. Butler

Due date: March 16, 2013

This assignment will have you creating a daemon to identify and regulate suspicious processes in the UNIX filesystem, in a way similar to how intrusion detection systems such as *Tripwire* operate. **Note the following closely:** Only students *not doing the course research project* need to submit this assignment.

1 Assignment Details

1. Create a tool called `phunt`, a UNIX daemon that searches for rogue processes and kills, suspends, or reduces the priority of them. The process behaviors to be tested are read from a blacklist. In order to do this assignment, you will need root access on a Linux system, and for that system to have the `/proc` filesystem enabled. (You will need to be root in order to read certain values out of `/proc`.) You should use the virtual machine image you used for Assignment 2 to do this.

2. The program will accept input from the command line that specifies its operation, as follows:

   ```
   phunt [-l <log file>] [-c <config file>] [-d]
   ```

   where the `-l` option specifies that log entries (described below) will be *appended* to the log file `<logfile>` and the `-c` option indicates that the configuration file (also described below) read by the program as startup should be `<config file>`. (Of course, these represent names to be input by the user.) The `-d` option is for extra credit, and indicates that the program should be run as a daemon (described below).

   If no log file is specified, the default of `/var/log/phunt.log` should be used. If no configuration file is specified, the default of `/etc/phunt.conf` should be used. If the logfile cannot be opened for append or the configuration file cannot be opened for reading, the program should abort and warn the user.

3. The program should log all meaningful information relating to its operation. For example, the program should log startup and shutdown events, scans of the system, actions of processes taken, etc. Each log entry should begin with a timestamp indicating when the event occurred. Look at `/var/log/syslog` for an example of what a timestamp on your system should look like. A sample log of the process starting up may include the following entries:

   ```
   Feb 21 14:08:12 ubuntu phunt:  phunt startup (PID=7642)
   Feb 21 14:08:13 ubuntu phunt:  opened logfile /var/log/phunt.log
   Feb 21 14:08:13 ubuntu phunt:  parsing configuration /etc/phunt
   ...
   ```

4. The configuration file read by the program lists blacklist behaviors to be monitored by the program. The file contains a list of commands, one per line, as follows:

   ```
   <action> <type> <param>
   ```

   Actions include the following:

   - `kill` – The process should be killed. Use the system `kill()` function to send a `SIGKILL` signal to the target process. The `phunt` program should check that the signal was successfully sent and confirm that the target program was actually killed.

- `suspend` – The process should be suspended. Use the system `kill()` function to send a `SIGTSTP` signal to the target process. The `phunt` program should check that the signal was successfully sent and confirm that the target program was actually suspended.
- `nice` – The process scheduling priority should be reduced to the lowest possible value (-20) using the system `nice()` function. The `phunt` program should confirm the new priority level.

Types include the following:

- `user <username>` – This behavior type indicates that any process run by a particular user `<username>` should be acted upon.
- `path <pathname>` – This behavior type checks the environment in which the process is running and determines whether it contains the search path `<pathname>`.
- `memory <MB>` – This behavior type determines whether the process memory usage exceeds `<MB>` megabytes.

All blank lines should be ignored, as should those containing a '#' character (indicating a comment line). The configuration information should be logged as it is parsed with some explanation regarding its meaning. The priority of the rules is implicit: rules appearing first in the configuration file have higher priority.

A sample configuration file follows:

```
# phunt.conf : phunt configuration file
# Last modified : Tue Feb 20 08:39:06 PST 2011

# dangerous users
kill user badguy

# nice users
nice user butler

# dangerous paths
suspend path /tmp

# memory ceiling
kill memory 1024
nice memory 512
```

5. Use the `/proc` filesystem to detect process behaviors. This filesystem exports system information in "virtual" files that can be read and written by processes. Each of the above behaviors will be detected by reading some file and parsing its contents. The system maintains a directory `/proc/<pid>/` for each process on the system, where `pid` is the process ID assigned to it by the system. Thus, scanning files in that directory will allow you to monitor its state. You will need to investigate the structure and contents of the `/proc` filesystem in more depth to complete this assignment.

6. *Extra credit:* You should daemonize the code. Daemons run in the background and are generally started by system startup scripts (e.g., in the `/etc/init` directory). Running the program in the background requires you to set up code to `fork()` the background process. You should create this code and enable its use when the `-d` option is given from the command line. A description of creating a Unix daemon can be found at `http://www.enderunix.org/docs/eng/daemon.php`. Note that you can look at the code on the site, but **do not** copy it. In particular, do not use the `daemonize()` function verbatim from this code. You must implement the fork code for the daemon yourself.

7. Create a gzipped tar file containing the commented code and the Makefile. Include a `README` file describing program operation and design decisions that you made to get the program running. The tarfile should be made in the same format as you used for assignment 1 (i.e., `<lastname>-assign3.tgz` that expands to a subdirectory `<lastname-assign3>` where your code is contained).

2 Developing Your Code

You will need to run Linux in order to perform this assignment. If you do not have a machine running Linux that you have root access on (in order to potentially install libraries as necessary), you should use a virtual machine. This also has the benefit that if you crash the system, it is straightforward to restart, and allows you to take snapshots of system state right before you do something potentially risky or hazardous, so that if something goes horribly awry you can easily roll back to a safe state. You may use the room 100 machines to run the Linux VM image within a Virtualbox environment, or run natively or within a VM on your own personal machine. Importantly, do not use `ix` for this assignment.

3 Submission Instructions

You **must** turn in the following:

1. A `README` file. The `README` will contain the following:
   - Your name
   - A list of submitted source files
   - Overview of work accomplished
   - Description of code and code layout
   - General comments and design decisions made, as well as anything else that can help us grade your code.
   - Number of hours spent on the project and level of effort required.
2. A `Makefile`. We should be able to compile your code by simply typing `make`. If you do not know what a `Makefile` is or how to write one, ask He Gu, the course GTF, or consult one of the many online tutorials.
3. The source code for your programs. These, along with your Makefile, should be well-commented.

Put all of your project files into their subdirectory called `hw4` and make sure that your `Makefile` has a "make clean" command to clear out any object files and other unneeded intermediate files. Run "make clean" on this subdirectory.

cd up a directory so that `hw4` is a subdirectory of your working directory, and run the following command to create your submission tarball, but replacing "UOEMAIL" with your cs.uoregon.edu email account name.

```
tar cvzf hw4_submission_UOEMAIL.tar.gz hw4}
```

For example, since my UO CIS email account is "butler", I would run the command:

```
tar cvzf hw4_submission_butler.tar.gz hw4}
```

Use the turnin script, located at `http://systems.cs.uoregon.edu/apps/turnin.cgi`, to submit your tarball.

4 Grading Guidelines

This programming assignment will be graded as follows:

- 10% Documentation
- 10% General code design
- 80% Functionality and correct operation of your program
- 10% bonus for daemonizing the code

Note that general deductions may occur for a variety of programming errors including memory violations, lack of error checking, poor code organization, etc. Also, do not take the documentation lightly, as it provides those evaluating your project with a general roadmap of your code; without good documentation, it can be quite difficult to grade the implementation. These assignments will be be graded on machines running the Ubuntu 11 Linux distribution.

The assignment is due at 11:59 PM Samoa Standard Time on **March 16, 2013.** You will have three grace days for late assignments for this class with the ability to use a maximum of two on any given assignment. You must specify that you intend to use these in your README file. If you exceed your allotment of grace days, points will be deducted at 25% per day. *Note that assignments will not be accepted after the final exam on March 19, 2013.*

5 Programming Note

A good overview of signal programming can be found at

`http://users.actcom.co.il/~choo/lupg/tutorials/signals/signals-programming.html`

There are other good resources including Stevens's "Advanced Programming in the UNIX Environment". In addition, you should (re)familiarize yourself with directory structures in Unix. I'd strongly recommend you understand the use of `getopt()` as well for parsing options from the command line.

6 Note on Academic Dishonesty

As with all assignments in this class, you are prohibited from copying any content from the Internet. You are also prohibited from sharing code, configurations, or text, or getting help with the basics of the assignment (using the library and developing your code) from anyone apart from the instructor and the GTF. This is an unfortunate necessity to cut down on cheating. Consulting online sources is acceptable, but under no circumstances should *anything* be copied.

We will be using MOSS to find similarities between the submitted project and those submitted by others in the class, so please do not be tempted to cheat.

Failure to abide by these requirements will result in academic sanctions up to dismissal from the class and involvement of Academic Affairs.