HeartSuite Setup and Configuration

HeartSuite server edition: Debian Linux base

x86 architecture, version 1

Karen Heart
karen@heartsecsuite.com
Heart Security Suite, LLC

Last modified: Nov. 14, 2023

Table of Contents

Introduction

HeartSuite, Server Edition, was developed as an aggressive weapon to combat malware. Specifically, it was designed to achieve Denial, a military term referring to actions that destroy an enemy's ability to wage war [1]. HeartSuite engages in preemptive strikes to achieve this goal. In particular, it prevents programs from starting unless they have been added to an Application Permission Orders (APO) record by an administrator using a HeartSuite tool. Moreover, HeartSuite prevents programs from accessing files unless and until permission orders for accessing those files have been added to the programs’ APO record.

Likewise, HeartSuite prevents programs from accessing remote computers unless and until permission orders for accessing their Internet addresses have been added to the programs’ APO record. Of course, attackers are notorious for convincing unsuspecting souls to download and install malicious updates to existing, useful programs. To deprive such Trojan malware from destroying files, HeartSuite automatically backs up all files designated for backup using another HeartSuite tool. Finally, the HeartSuite version manager tool can be used later to restore any version of a backed-up file, regardless of whether it was encrypted, deleted, or otherwise modified by malware. In short, HeartSuite eliminates malware by disabling its functionality, even before it can be detected.

Overview of Setup

Immediately after it is installed, HeartSuite is not yet ready to enter Denial mode due to the fact that HeartSuite neither discriminates between programs nor considers user privileges. It prevents necessary and useful startup programs, as well as shutdown programs, from being executed because no APO record exists for such programs. Accordingly, these programs must first be identified and added to an APO record in the HeartSuite database, along with their preferred access privileges, before HeartSuite can deploy a practical defense without interfering with legitimate operations.

Consequently, HeartSuite must be launched initially in Monitor mode. In monitor mode, HeartSuite merely logs every action that it, otherwise, would block in Denial mode. Thus, Monitor mode serves as the reconnaissance needed for staging future Denial actions. Nonetheless, HeartSuite version backup also operates in this mode. Hence, even if one is merely watching the activity of a server using Monitor mode, HeartSuite provides some level of assurance through use of continuous automated backups.

To assure that HeartSuite is launched in this mode, the installation routine sets Monitor mode as the default. Once the configuration of permission orders records has been completed, an administrative user uses another HeartSuite tool, described below, to change the mode to Denial. **In short, HeartSuite setup must be completed before entering Denial mode; otherwise, your computer will no longer boot or shutdown!**

[1] See “List of military strategies and concepts”, Wikipedia, https://en.wikipedia.org/wiki/List_of_military_strategies_and_concepts

System Requirements

This edition of HeartSuite was compiled on a Debian 11 server, running on an x86 chipset. Therefore, it is binary compatible with any Linux distro derived from Debian or Ubuntu running on an x86 chipset. Additionally, HeartSuite is distributed as a set of tools and a gingerly modified version of the mainline Linux kernel, specifically version 5.19.6. Accordingly, this modified kernel must be booted in order to utilize HeartSuite.

Obtaining HeartSuite

The beta version is distributed as a single tar file; download it from our website, `heartsecsuite.com`. Please note that access to the tar file via `wget` is turned off by our hosting provider; you must use the download form on our website.

Installation

Preliminary Step: Virtual Machines (VM's) on Clouds

Installation of HeartSuite requires rebooting your computer several times. Notably, our installation procedure utilizes the GRUB boot loader, which can use labels or UUID's when booting. We have found that use of labels for deployments in a cloud can result in boot failures. Accordingly, we strongly recommend that you edit the GRUB settings for VM's provisioned in the cloud. Specifically, we recommend that you comment out the following line in the `/etc/default/grub` file:

```
GRUB_DISABLE_LINUX_UUID=true
```

by preceding it with a pound (#) symbol:

```
#GRUB_DISABLE_LINUX_UUID=true
```

Then, rebuild GRUB using `/sbin/update-grub`.

Installing HeartSuite – Part 1

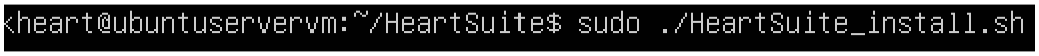
Untar the distribution tar file:

```
# tar xvf 5.19.6-HeartSuite-1.0.tar -m
```

```
kheart@ubuntuservervm:~/HeartSuite$ tar xvf 5.19.6-HeartSuite-1.0.tar -m
```

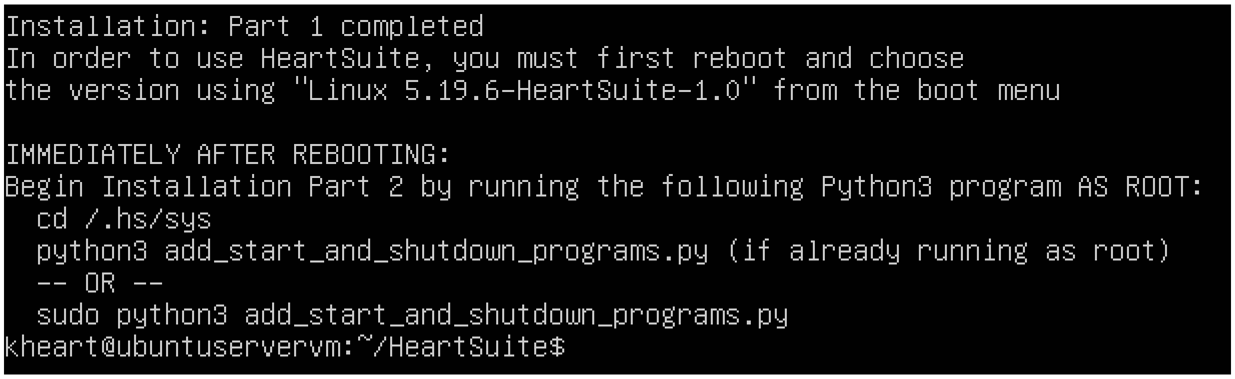
Run the installation script **as root**:

```
# sudo ./HeartSuite_install.sh
```
--OR--
```
# su  # if you're not logged in as root
# ./HeartSuite_install.sh
```



```
kheart@ubuntuservervm:~/HeartSuite$ sudo ./HeartSuite_install.sh
```

After performing this part of the installation, the script will display the following message:

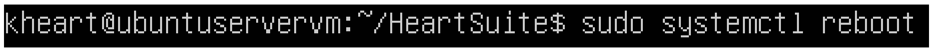


```
Installation: Part 1 completed
In order to use HeartSuite, you must first reboot and choose
the version using "Linux 5.19.6-HeartSuite-1.0" from the boot menu

IMMEDIATELY AFTER REBOOTING:
Begin Installation Part 2 by running the following Python3 program AS ROOT:
  cd /.hs/sys
  python3 add_start_and_shutdown_programs.py (if already running as root)
  -- OR --
  sudo python3 add_start_and_shutdown_programs.py
kheart@ubuntuservervm:~/HeartSuite$
```

AT THIS POINT IN THE INSTALLATION PHASE, YOU MUST REBOOT IN ORDER FOR THE HEARTSUITE-MODIFIED KERNEL TO BE LOADED!

Use the systemctl command to reboot:

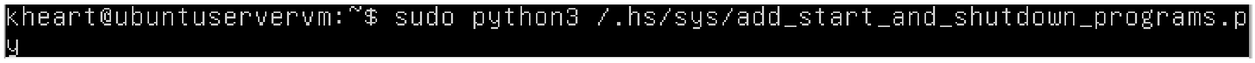


```
kheart@ubuntuservervm:~/HeartSuite$ sudo systemctl reboot
```

Installing HeartSuite – Part 2

You must augment the initial HeartSuite APO database so that essential startup and shutdown programs will be permitted to run as needed. A Python script named `add_start_and_shutdown_programs.py` is included for this purpose. It must be run **as root** using Python3:

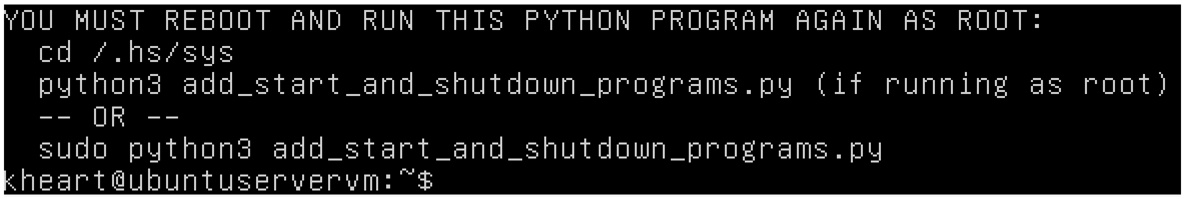
```
# sudo python3 /.hs/sys/add_start_and_shutdown_programs.py
     --OR--
# su  # if you're not logged in as root
# python3 /.hs/sys/add_start_and_shutdown_programs.py
```



```
kheart@ubuntuservervm:~$ sudo python3 /.hs/sys/add_start_and_shutdown_programs.p
y
```

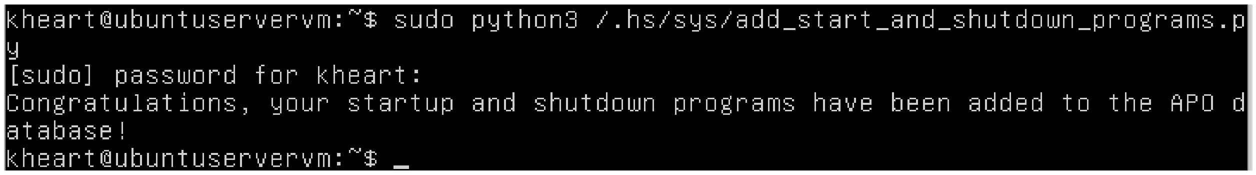
Each time you run this script, it will scan the HeartSuite log, described below, and the kernel log for permission error messages generated by HeartSuite. The script will then use these permission messages to run a HeartSuite tool named `hs-app-perm-orders-manager` to build appropriate APO records.

After building these records, you must reboot and then run the script again because new permission error messages will likely be generated. In fact, the script will direct you to reboot and run it again with the following message:



```
YOU MUST REBOOT AND RUN THIS PYTHON PROGRAM AGAIN AS ROOT:
  cd /.hs/sys
  python3 add_start_and_shutdown_programs.py (if running as root)
  -- OR --
  sudo python3 add_start_and_shutdown_programs.py
kheart@ubuntuservervm:~$
```

After three cycles of rebooting and running the script, the APO database *should* be complete for startup and shutdown programs. If so, the script will print the message, “Congratulations, your startup and shutdown programs have been added to the APO database!”



```
kheart@ubuntuservervm:~$ sudo python3 /.hs/sys/add_start_and_shutdown_programs.p
y
[sudo] password for kheart:
Congratulations, your startup and shutdown programs have been added to the APO d
atabase!
kheart@ubuntuservervm:~$ _
```

Depending on your distro, it may take four or five cycles of rebooting and running the script in order to complete this part of the Installation.

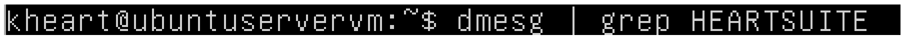
Note that the `add_start_and_shutdown_programs.py` script also writes certain messages to standard output. As you run the this script, you will see error messages directed to standard error by the underlying calls to `hs-app-perm-orders-manager`. The `add_start_and_shutdown_programs.py` script also adds its own messages but to standard out. These messages indicate when permission is substituted to access a directory rather than a file within it. By default, the `add_start_and_shutdown_programs.py` script adds directory permission instead of file permission whenever it detects that a particular file may be temporary in nature. The script assesses this quality based on whether the name of the file contains digits or capital letters. Additionally, the `add_start_and_shutdown_programs.py` script evaluates directory names that contain digits as temporary, as well. You can change this assessment by adding text to the list of exceptions in the script; please view the script, starting at Line 34, for more information and examples.

Repeating these cycles is vital before changing to Denial mode. If you fail to setup HeartSuite properly using the `add_start_and_shutdown_programs.py` program multiple times, your computer will not boot or shutdown when you change to Denial mode; instead, it will merely hang!

Verifying that HeartSuite is Running

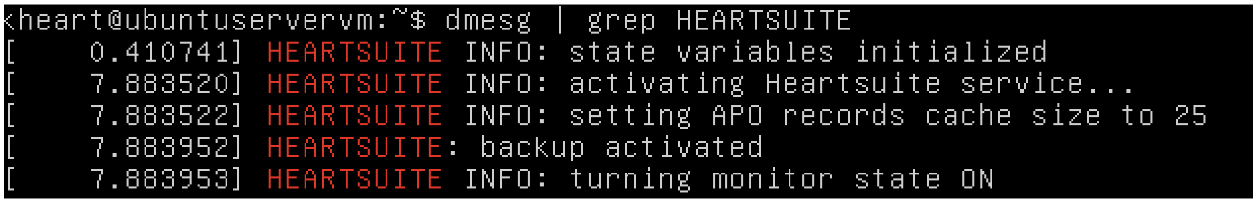
To confirm that HeartSuite is running, view the kernel log messages using the following command, which may require root permission:

```
# dmesg | grep HEARTSUITE
```



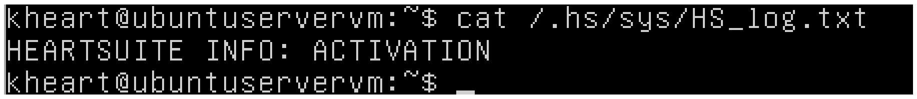
```
kheart@ubuntuservervm:~$ dmesg | grep HEARTSUITE
```

If HeartSuite is running, there will be at least two informational messages displayed in the log, one indicating that HeartSuite has been activated and another indicating its operation mode, Monitor or Denial.



```
kheart@ubuntuservervm:~$ dmesg | grep HEARTSUITE
[    0.410741] HEARTSUITE INFO: state variables initialized
[    7.883520] HEARTSUITE INFO: activating Heartsuite service...
[    7.883522] HEARTSUITE INFO: setting APO records cache size to 25
[    7.883952] HEARTSUITE: backup activated
[    7.883953] HEARTSUITE INFO: turning monitor state ON
```

If HeartSuite was not running when activated, a message indicating the size of the APO records cache will also be added to the kernel log. A simple activation message will also be added to the HeartSuite log file:



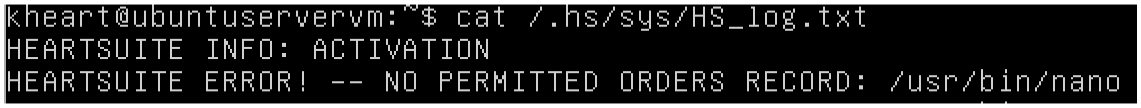
```
kheart@ubuntuservervm:~$ cat /.hs/sys/HS_log.txt
HEARTSUITE INFO: ACTIVATION
kheart@ubuntuservervm:~$ _
```

Setting Up Executable Programs After Installation

Introduction to using the HeartSuite log

While running, HeartSuite always attempts to capture permission errors to the HeartSuite log, `/.hs/sys/HS_log.txt`. Initially, it will report programs that are executed and that do not have an APO record. For example, if you run the `nano` program before adding it to an APO record, the HeartSuite log will contain the following entry:

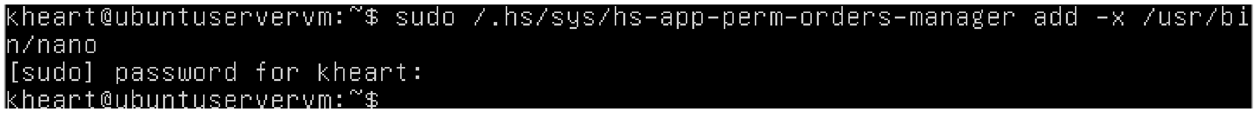
```
HEARTSUITE ERROR! -- NO PERMITTED ORDERS RECORD: /usr/bin/nano
```



```
kheart@ubuntuservervm:~$ cat /.hs/sys/HS_log.txt
HEARTSUITE INFO: ACTIVATION
HEARTSUITE ERROR! -- NO PERMITTED ORDERS RECORD: /usr/bin/nano
```

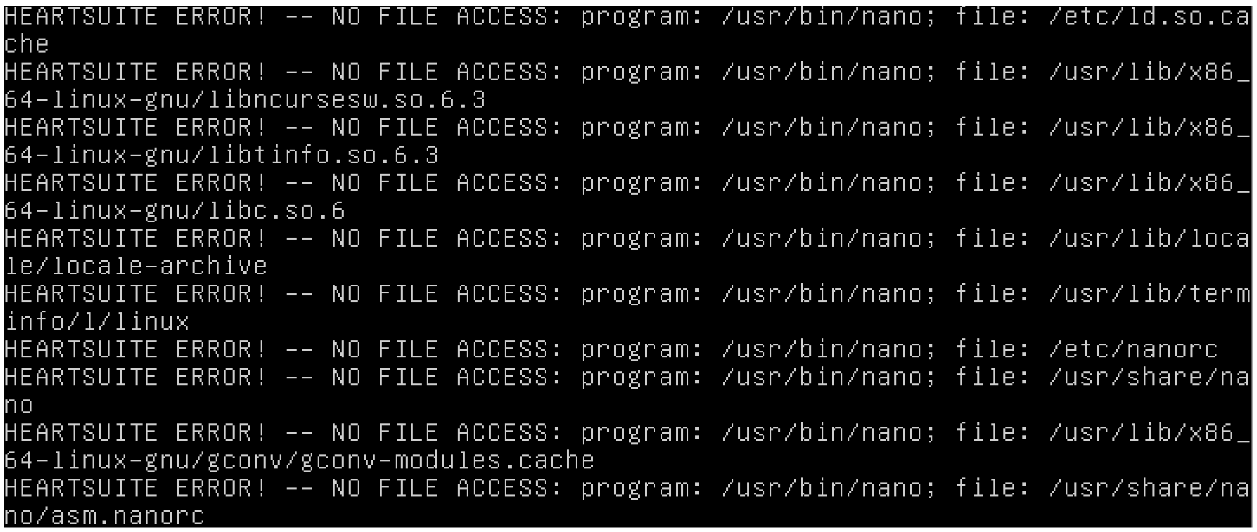
After observing this message, you can create an APO record for the `nano` program manually, **as root**, as follows:

```
# sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano
```
--OR, IF RUNNING AS ROOT--
```
# /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bi
n/nano
[sudo] password for kheart:
kheart@ubuntuservervm:~$
```

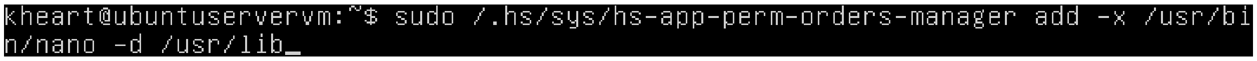
If you then run `nano` again, HeartSuite will start reporting access permission errors for all files that `nano` accesses and the IP address of any remote servers that it connects to. Please note that `nano`, like most programs, relies on shared libraries to execute; accordingly, those libraries will be listed in the HeartSuite log as files accessed without permission. For example, after creating the APO record for `nano` on a Debian server and running it again, the following file access permission errors were generated:



```
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /etc/ld.so.ca
che
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/x86_
64-linux-gnu/libncursesw.so.6.3
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/x86_
64-linux-gnu/libtinfo.so.6.3
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/x86_
64-linux-gnu/libc.so.6
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/loca
le/locale-archive
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/term
info/l/linux
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /etc/nanorc
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/share/na
no
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/lib/x86_
64-linux-gnu/gconv/gconv-modules.cache
HEARTSUITE ERROR! -- NO FILE ACCESS: program: /usr/bin/nano; file: /usr/share/na
no/asm.nanorc
```

In fact, additional file access permission error messages were generated, but they are not shown in the example for the sake of brevity. In order to grant `nano` permission orders to access those files, you must add file permission orders to the APO record for `nano`. Specifically, you must add either the directories in which the files are located or the complete file paths themselves. In fact, you can include one directory or file path at the time the APO record for `nano` is created. For example, to add the directory `/usr/lib` to the APO record when it is created, you would use a modified version of the prior command:

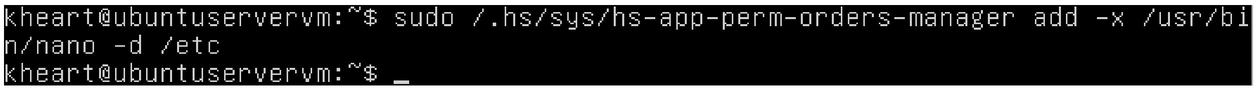
```
# sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano -d
/usr/lib
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bi
n/nano -d /usr/lib_
```

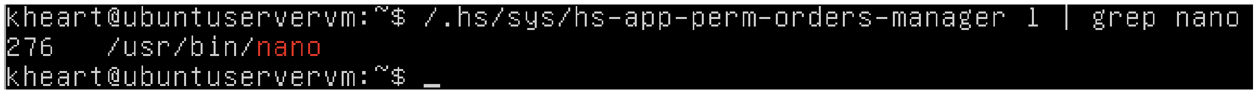
Thereafter, you can add more directories and file paths to the `nano` APO record in two ways. One way uses the program name, the other uses the program's APO record number. For example, to add the `/etc` directory to the `nano` APO record, use the following command after the `nano` APO record has been created:

```
# sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano -d
/etc
```



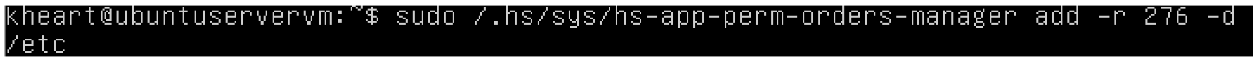
```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bi
n/nano -d /etc
kheart@ubuntuservervm:~$ _
```

Otherwise, you can extract the APO record number using the list command, coupled with a simple `grep`:


```
kheart@ubuntuservervm:~$ /.hs/sys/hs-app-perm-orders-manager l | grep nano
276   /usr/bin/nano
kheart@ubuntuservervm:~$ _
```

In this example, because HeartSuite has assigned the temporary record number 276 to the `nano` APO record, you can use the record number as follows to add the `/etc` directory:

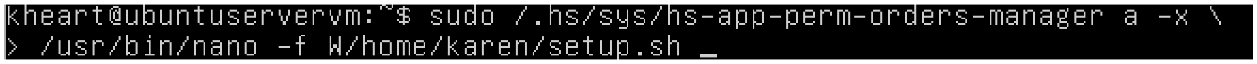
```
# /.hs/sys/hs-app-perm-orders-manager add -r 276 -d /etc
```


```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager add -r 276 -d
/etc
```

NOTE: As you remove APO records, APO record numbers will change! Accordingly, you must ensure that you are using the current record number before modifying an APO record using its number!

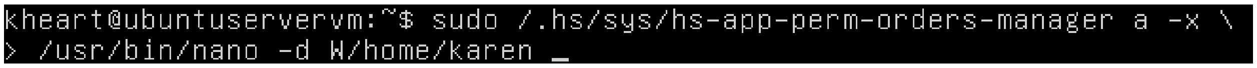
Notably, all of these examples grant read-only file access to files. In order to grant write permission (which includes read permission, as well) you must prefix either the directory name or file path with a capital ‘W’. For example, to permit the `nano` program to write to a file named `setup.sh` in the home directory of user ‘`karen`’, use the following command:

```
# sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano \
 -f W/home/karen/setup.sh
```


```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager a -x \
> /usr/bin/nano -f W/home/karen/setup.sh _
```

To grant `nano` permission to write to all files in `karen`’s home directory, use this command instead:

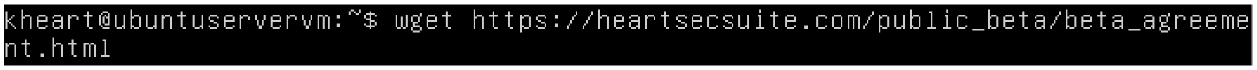
```
# /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/nano \
 -d W/home/karen
```


```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager a -x \
> /usr/bin/nano -d W/home/karen _
```

The same procedures apply to learning which remote servers a program attempts to access, as well as permitting the program to access particular remote servers. This edition of HeartSuite is strictly limited to specific IPv4 and IPv6 addresses, it cannot regulate network access using address ranges or domain names. Address ranges and domain name-based access, however, are planned for later editions.

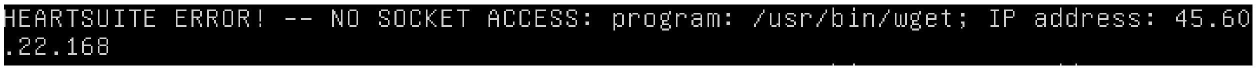
For example, suppose that an APO record is created for the `wget` program without adding any network permission orders. Then, the program is used to obtain an HTML document from a website, as follows:

```
# wget https://heartsecsuite.com/public_beta/beta_agreement.html
```


```
kheart@ubuntuservervm:~$ wget https://heartsecsuite.com/public_beta/beta_agreeme
nt.html
```

The following permission error message concerning network access appears in the log file:

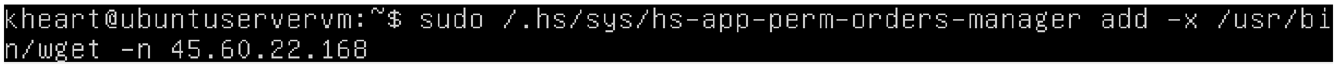
```
HEARTSUITE ERROR! -- NO SOCKET ACCESS: program: /usr/bin/wget; IP
address: 45.60.22.168
```



```
HEARTSUITE ERROR! -- NO SOCKET ACCESS: program: /usr/bin/wget; IP address: 45.60
.22.168
```

You can add this IP address to the APO record for `wget` as follows:

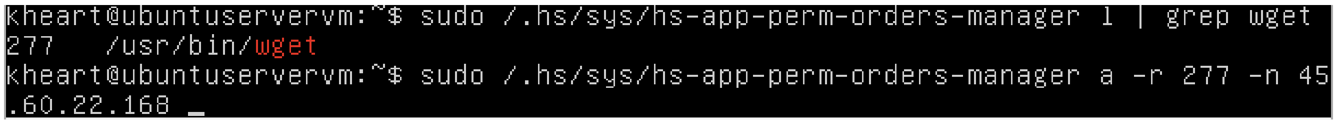
```
# sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bin/wget -n
45.60.22.168
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager add -x /usr/bi
n/wget -n 45.60.22.168
```

– or –

```
# /.hs/sys/hs-app-perm-orders-manager list | grep wget
277   /usr/bin/wget
# /.hs/sys/hs-app-perm-orders-manager add -r 277 -n 192.142.166.196
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager l | grep wget
277   /usr/bin/wget
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-app-perm-orders-manager a -r 277 -n 45
.60.22.168 _
```

The next time you run the same `wget` command, there will be no permission error message in the log for this IP address access.

Introduction to using the kernel log

Depending on you distro, some permission error messages may not appear in the HeartSuite log; instead, they are placed in the kernel log. You can easily obtain the kernel log with the following command, which may require root privilege:

```
# dmesg
```

```
kheart@ubuntuservervm:~$ sudo dmesg
```

In order to readily extract only the HeartSuite messages, use `grep`:

```
# dmesg | grep HEARTSUITE
```

```
kheart@ubuntuservervm:~$ dmesg | grep HEARTSUITE
```

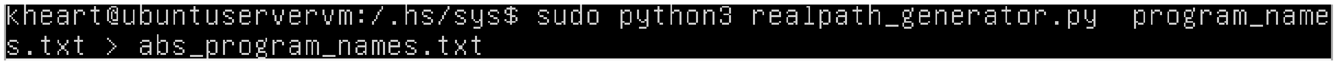
You can use the `add_start_and_shutdown_programs.py` script or just add permission orders manually, as described above.

Please note that any program added to the APO database using the `add_start_and_shutdown_programs.py` script is granted a permission order allowing it to access every file, or directory, specified in log messages. **Therefore, it is strongly recommended that you use the `add_start_and_shutdown_programs.py` script cautiously. Specifically, we recommend that you use it before any adding programs that do not come with your distro. We have found that it can take several days, possibly a week, to capture all of the relevant log messages involving processes run by systemd timers and cron jobs. Once this essential information is captured, we strongly recommend that you switch to other means. Primarily, we strongly recommend that you add permission orders for other programs manually so that you can tailor their permission orders to a "need-to-access" basis only. If you have several programs that require similar permissions, you can use one of our other scripts, which you can also modify easily for your needs.** Finally, you can use the `hs-app-perm-orders-manager` tool to review the access permissions of all programs and restrict them on the basis of need-to-access.

Other tools for adding records in batch

A few other tools are included to aid in creating APO records in batch. Because you need the absolute path to a program in order to create its APO record, a Python script named `realpath_generator.py` has been included for this task. This script accepts the name of a file containing names of programs, one per line; it then outputs their absolute paths if they are found using the `PATH` variable. You can capture the output by redirecting it to a file, as in this example:

```
# python3 realpath_generator.py program_names.txt >
    abs_program_names.txt
```



```
kheart@ubuntuservervm:/.hs/sys$ sudo python3 realpath_generator.py  program_name
s.txt > abs_program_names.txt
```

You can then use one of the batch scripts to create the APO records. If a program is not found by this Python script because it is not located with a path found in the `PATH` variable, you must find the program's path manually.

The `batch_record_add_root.py` script will create new APO records that give the programs a permission order to access to any file for reading. A more secure choice would be the `batch_record_add.py` script, which adds only limited file permission orders that are generally needed for a program to execute. Specifically, this latter Python script adds only read access for the `/usr/lib` and `/etc` directories because, on Debian-based systems, programs routinely rely on files contained in those directories when they start. By using this more limited APO record, you can watch for permission error messages and slowly add needed file access permission orders.

Please note that the `/.hs/sys` directory is owned by root; therefore, you can experience problems when creating files in that directory while using `sudo`. The easiest solution is to start a `sudo` shell, using the following command:

```
# sudo -s
```

```
kheart@ubuntuservervm:~$ sudo -s_
```

You can then run any command as root. You can exit that shell by pressing `Ctrl-d`.

Also, because error messages can quickly accumulate in the HeartSuite log, a simple utility permits you to easily clear the log. Run the following command **as root** to clear the log:

```
# /.hs/sys/empty_HS_log.sh
```

```
root@ubuntuservervm:/home/kheart# /.hs/sys/empty_HS_log.sh
```

Further, by leaving your server running continuously, the HeartSuite and kernel logs will eventually capture information about the programs executed in conjunction with `systemd` timers and `cron` jobs, as mentioned above. Again, use the same procedure to add them to the HeartSuite APO database.

Configuring use of HeartSuite shim programs

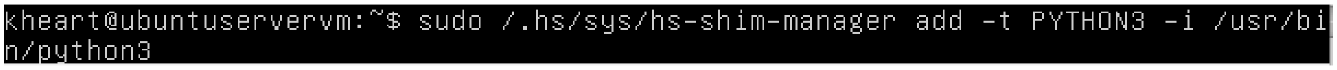
Certain programs are designed as language interpreters and characteristically operate on files containing interpreted code. Most of these programs, including Python, PHP, and Bash, expect source code files. Others, like Java, expect bytecode translations of source code; they may directly interpret the bytecode or compile it to native binary form for execution by the CPU. For purposes of discussion, all such programs will be referred to as interpreter programs.

If the APO record for an interpreter program were responsible for all of the potential permission orders that might be required by the various interpreted code files written for that interpreter program, it would likely have to provide wide latitude in permission orders, regardless of the purpose of any particular interpreted code file. Thus, limiting permission orders of all interpreted code files to a single APO record associated with the interpreter program would prove infeasible for maintaining security; in fact, that approach would provide an attack surface. Accordingly, HeartSuite permits admins to provide APO records for interpreted code files, thereby permitting an admin to circumscribe the actions that particular code files can perform. Indeed, HeartSuite permits an admin to remove the APO record of an interpreter program without disturbing the APO records of interpreted code files. In that case, only approved interpreted code files can be executed. Notably, HeartSuite treats interpreted code files exactly the same as all other ordinary binary programs.

Because interpreter programs have no knowledge of HeartSuite though, HeartSuite includes additional tools, known as shim programs, that are used for executing interpreted code files. Specifically, it includes a small shim program for launching each associated interpreter program. For example, HeartSuite includes a shim program named `hs_python3`, which launches the Python 3 interpreter program. By first parsing the command line however, `hs_python3` identifies the interpreted code file that is to be executed and applies its APO record, rather than the APO record, if any, belonging to the Python 3 interpreter program. Hence, enforcement of application permission orders for interpreted code files can only be achieved by use of HeartSuite shim programs.

There are two approaches for using shim programs. The direct case is to configure the system so that they must be used directly, instead of their associated interpreter programs. To configure the file name of the interpreter program that will be launched by a shim program, use the `hs-shim-manager` program. For example, suppose that your server contains a symbolic link, `python3`, that points to the executable file, `python3.9`; both reside in the `/usr/bin` directory. To point `hs_python3` to Python3, add the following record to the shim configuration database, which points `hs_python3` to the Python3 symbolic link:

```
# /.hs/sys/hs-shim-manager add -t PYTHON3 -i /usr/bin/python3
```

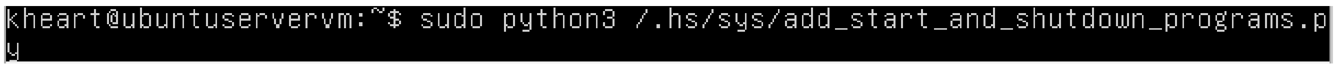


```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-shim-manager add -t PYTHON3 -i /usr/bi
n/python3
```

In that case, Python scripts launched with `hs_python3` would use the script's APO record, while scripts launched directly with `python3` would use the `python3.9` APO record.

As shown above, the `add_start_and_shutdown_programs.py` script is started using Python3 directly:

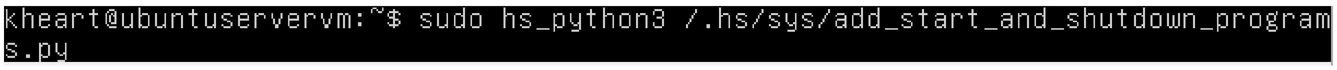
```
# python3 /.hs/sys/add_start_and_shutdown_programs.py
```



```
kheart@ubuntuservervm:~$ sudo python3 /.hs/sys/add_start_and_shutdown_programs.p
y
```

To use the Python3 shim program, use the shim program name instead of the Python3 link:

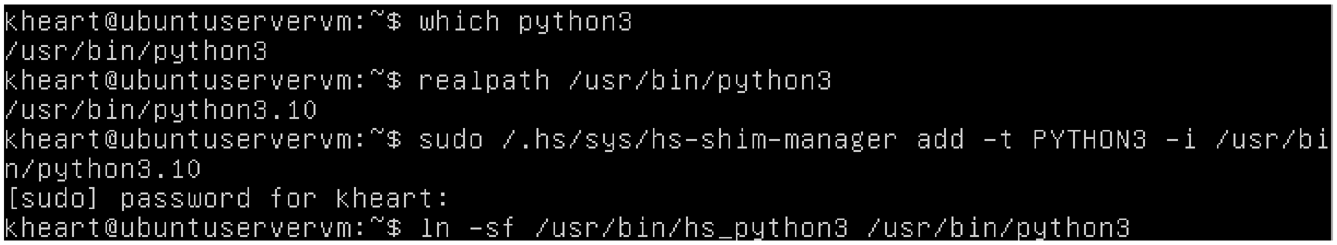
```
# hs_python3 /.hs/sys/add_start_and_shutdown_programs.py
```



```
kheart@ubuntuservervm:~$ sudo hs_python3 /.hs/sys/add_start_and_shutdown_program
s.py
```

The other approach is to point the symbolic link for the interpreter program to the shim program. This approach is usually possible because interpreter program names are typically just symbolic links. For example, to compel users to always use the shim program for Python3, configure the shim program to point to the executable file and redirect the `python3` link to point to the shim program, as follows:

```
# /.hs/sys/hs-shim-manager add -t PYTHON3 -i /usr/bin/python3.10
# ln -sf /usr/bin/hs_python3 /usr/bin/python3
```



```
kheart@ubuntuservervm:~$ which python3
/usr/bin/python3
kheart@ubuntuservervm:~$ realpath /usr/bin/python3
/usr/bin/python3.10
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-shim-manager add -t PYTHON3 -i /usr/bi
n/python3.10
[sudo] password for kheart:
kheart@ubuntuservervm:~$ ln -sf /usr/bin/hs_python3 /usr/bin/python3
```

In this latter situation, any Python script launched using the `python3` link would first launch `hs_python3`, which would then launch `python3.10`. `hs_python3` would select the correct APO record to apply. This edition of HeartSuite includes shim programs for Python 3, Python 2, Perl, and PHP. A shim program for Java is planned for later editions.

Configuring Automatic File Backup

HeartSuite uses a separate database of directories to determine whether to backup a file that has been written. By default, the backup configuration database includes only a single directory, `/home`. You can remove it, as well as add additional directories, by using the `hs-backup-config-manager` tool. HeartSuite will automatically backup each file in these specified directories, including those within their subdirectories, when the contents of the file are changed by writing.

Unless you have unlimited storage space, you must consider carefully which directories are to be included in the backup. For example, while there may be reason to backup system logs found in /var, backing up each version of a log file will consume a great amount of storage, thereby filling up mounted disk space quickly. Please consider limiting backup to directories that contain user documents and executable files, including shared libraries. Inclusion of executables will easily permit you to roll back programs in the unfortunate case of a malicious update attack.

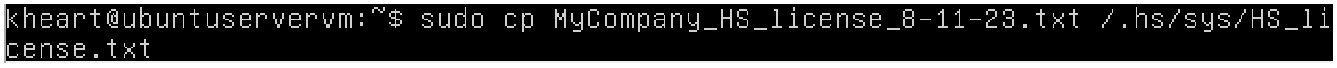
Switching to Denial Mode and Licensing

Licensing

In order to switch from Monitor to Denial mode, you need a license for your server; you must also register the server using the license. The license is a simple text file; however, one license can potentially be used to register up to 9999 servers. At the time you purchase your license, you must specify how many servers you want to be covered by the license. You can purchase additional licenses, if needed.

After you have downloaded the license file, you must copy it to each server that it covers. Regardless of the name of your license file, you must copy it to each server as, “`HS_license.txt`” in the `/.hs/sys` directory. For example, if you named your license file, “`MyCompany_HS_license_8-11-23.txt`”, you would copy it using the following command:

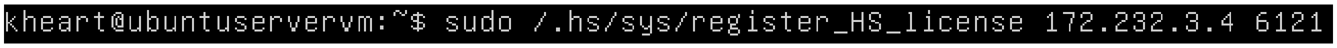
```
# sudo cp MyCompany_HS_license_8-11-23.txt /.hs/sys/HS_license.txt
```



```
kheart@ubuntuservervm:~$ sudo cp MyCompany_HS_license_8-11-23.txt /.hs/sys/HS_li
cense.txt
```

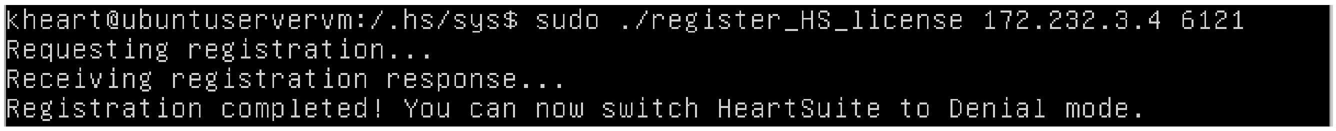
After you copy the license, you must register the license by using the `register_HS_license` program. The command line arguments needed are the IP address of the HeartSuite Registration Server and the port number, which is 6121. At the time of this writing, the IP of the HeartSuite Registration Server is 172.232.3.4. Please check our website to learn whether the IP and/or port number has changed. Using this data, you would run the registration program as follows:

```
# sudo /.hs/sys/register_HS_license 172.232.3.4 6121
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/register_HS_license 172.232.3.4 6121
```

If registration is successful, the program will create a registration key that is used by another HeartSuite tool and display an appropriate message.



```
kheart@ubuntuservervm:/.hs/sys$ sudo ./register_HS_license 172.232.3.4 6121
Requesting registration...
Receiving registration response...
Registration completed! You can now switch HeartSuite to Denial mode.
```

If an error occurs, the registration program will display an error message. You need to register your server only once.

Monitor vs Denial mode

At some point, you need to switch to Denial mode in order to prevent malicious programs from starting, or at least to restrict the files and remote computers such programs may access. You activate Denial mode using the `hs-monitor-state` program, as explained below. Before switching to Denial mode however, you must have successfully run the `add_start_and_shutdown_programs.py` script, as explained above, repeatedly until you have received the message, “Congratulations, your startup and shutdown programs have been added to the APO database!”. Characteristically, the message will appear after running the program three or four times.

As stated above, if you fail to setup HeartSuite properly using the `add_start_and_shutdown_programs.py` program multiple times, your computer will not boot or shutdown when you change to Denial mode; instead, it will merely hang!

Moreover, if you fail to add needed file permission orders or network address permission orders to APO records, HeartSuite will actively block programs from accessing needed files and network addresses when you change to Denial mode.

Once HeartSuite has been configured as desired, it would behoove you to continue using it in Monitor mode for some time, such as a few days or even a week or so. During that time, you will be able to use the logs to gather more information about file and network access activity. This information could prove invaluable for further APO record configuration before moving to Denial mode.

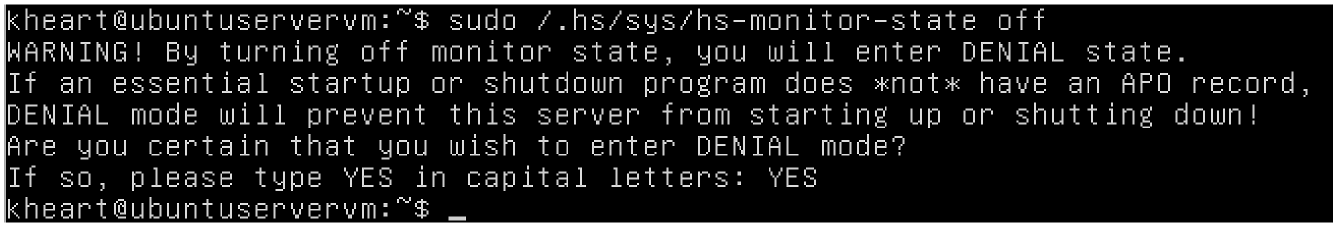
Please note that, at times, you must revert to Monitor mode when installing new software. For example, the Debian package manager, `dpkg`, creates temporary directories when installing packages. In Denial mode, this behavior will generate a permission error and cause the program to halt. By the time you have seen the error however, the temporary directory has been removed; therefore, it cannot be added to an APO record. Thus, the simple solution is to use `dpkg` in Monitor mode, then add any additional permission orders needed, then return to Denial mode.

Switching Modes

To switch to Denial mode, you instruct the `hs-monitor-state` program to turn monitoring off, which is how Denial mode is activated:

```
# sudo /.hs/sys/hs-monitor-state off
```

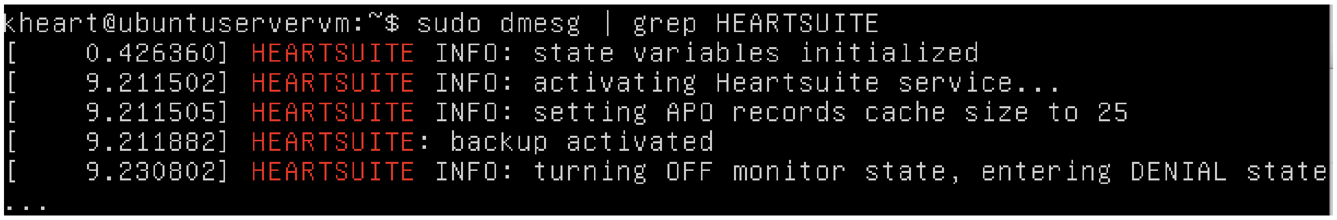
This program will verify that you, in fact, wish to turn off Monitoring and turn on Denial mode. Specifically, it will display a warning message and require that you signify your intent to activate Denial mode by typing the word ‘YES’, in all capital letters:



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-monitor-state off
WARNING! By turning off monitor state, you will enter DENIAL state.
If an essential startup or shutdown program does *not* have an APO record,
DENIAL mode will prevent this server from starting up or shutting down!
Are you certain that you wish to enter DENIAL mode?
If so, please type YES in capital letters: YES
kheart@ubuntuservervm:~$ _
```

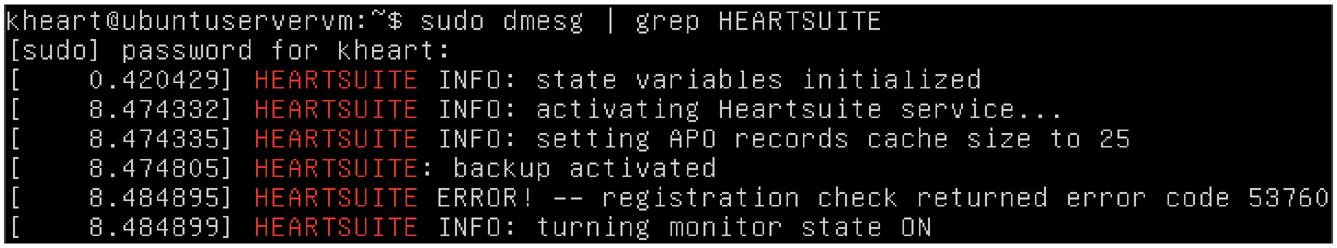
After turning off monitoring mode, you must then either reactivate HeartSuite, by running the activate_HS program, or reboot. This final step is necessary for Denial mode to be activated. Thereafter, HeartSuite will always boot in Denial mode until you use the `hs-monitor-state` program to change back to Monitor mode, which you must do before installing packages.

You can view the result of trying to activate Denial mode by reading the kernel log:



```
kheart@ubuntuservervm:~$ sudo dmesg | grep HEARTSUITE
[    0.426360] HEARTSUITE INFO: state variables initialized
[    9.211502] HEARTSUITE INFO: activating Heartsuite service...
[    9.211505] HEARTSUITE INFO: setting APO records cache size to 25
[    9.211882] HEARTSUITE: backup activated
[    9.230802] HEARTSUITE INFO: turning OFF monitor state, entering DENIAL state
...
```

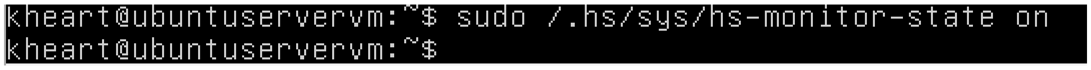
If a problem occurs, you can view the error condition in the kernel log:



```
kheart@ubuntuservervm:~$ sudo dmesg | grep HEARTSUITE
[sudo] password for kheart:
[    0.420429] HEARTSUITE INFO: state variables initialized
[    8.474332] HEARTSUITE INFO: activating Heartsuite service...
[    8.474335] HEARTSUITE INFO: setting APO records cache size to 25
[    8.474805] HEARTSUITE: backup activated
[    8.484895] HEARTSUITE ERROR! -- registration check returned error code 53760
[    8.484899] HEARTSUITE INFO: turning monitor state ON
```

You may switch back to Monitor mode anytime by using the `hs-monitor-state` program again:

```
# sudo /.hs/sys/hs-monitor-state on
```



```
kheart@ubuntuservervm:~$ sudo /.hs/sys/hs-monitor-state on
kheart@ubuntuservervm:~$
```

Lockdown: Protecting Denial Mode

The `lockdown_HS` program protects HeartSuite settings during Denial mode. In particular, HeartSuite prevents any changes to the APO records and other settings once lockdown is initiated.

Moreover, the `HS_lockdown.sh` script starts the `lockdown_HS` program and also protects others files and directories from tampering by attackers during lockdown. Technically speaking, lockdown lasts until the next time your server is booted; there is no direct way to turn lockdown off. Please note that Lockdown cannot be engaged in Monitor mode; if you try to do so, an error message is merely written to the kernel log.

Included in the HeartSuite installation is a `systemd` service unit named `heartsuite.service`. This service unit executes a shell script named `HS_startup.sh` at startup. The `HS_startup.sh` script executes the `activate_HS` program, which is the program that actually activates HeartSuite. Thus, this sequence of events starts HeartSuite automatically after booting. The `HS_startup.sh` script also contains a line that invokes the `HS_lockdown.sh` script immediately after activating HeartSuite, but, by default, the line is commented out. In order to engage lockdown immediately after booting in Denial mode, you must first uncomment this line. Once the line is uncommented, the startup script will always call the lockdown script. In that situation, rebooting will always engage lockdown before you can prevent it. In order to disengage lockdown, you must boot to an alternate kernel; this procedure will be discussed in the section below, "Protecting Your Server During Maintenance."

The essence of lockdown involves protecting files and directories from being changed. This protection is achieved by first making a file or directory immutable. Files and directories can be made immutable programmatically or by use of the command line tool, `chattr`. You can view whether particular files and directories are immutable using the `lsattr` command line tool. The `chattr` tool can also be used to make files and directories mutable again. Lockdown merely disables `chattr` functionality; therefore, no changes in mutability can be made once lockdown is engaged.

Although the `lockdown_HS` program supplied with HeartSuite can be executed directly, we urge you to run it indirectly. Specifically, we strongly recommend that you execute the `HS_lockdown.sh` script. This script provides a list of files and directories that you should consider protecting with lockdown. The script makes files and directories immutable using `chattr`, then executes the `lockdown_HS` program. Please feel free to edit this script to meet your needs, but keep in mind that once the `lockdown_HS` program has been executed, no changes in mutability may be made.

Files and directories may be made mutable again once lockdown is no longer active. The program `hs-unlock-progs`, also supplied with HeartSuite, switches all HeartSuite files back to being mutable so that you can make further configuration changes. The script `HS_unlock.sh` runs the `hs-unlock-progs` program and then makes other files and directories mutable again. In essence, this script is the counterpart to the `HS_lockdown.sh` script. Thus, any files or directories that are made immutable with the `HS_lockdown.sh` script should be returned to a mutable state using the `HS_unlock.sh` script before making changes to your server. If you forget to run the `HS_unlock.sh` script after lockdown ends and then try to write to a file made immutable by the `HS_lockdown.sh` script, you will encounter the error, "`could not open <filename> file; errno:1.`" The fix is simple: run the `HS_unlock.sh` script, then try your program again.

Preventing Gaps in Your HeartSuite Configuration

Undoubtedly, you will add APO records for quite a few programs. Many of these programs are useful primarily during maintenance, such as for editing files or moving them around. Typically, server programs have little use for such programs during operation. Therefore, there is no reason why such programs should remain executable during lockdown. After all, if an attacker can gain access to a shell through a vulnerability in one of your server programs, the attacker could potentially cause havoc by using such programs.

For example, the `rm` program simply deletes files, which is very useful during maintenance, potentially throughout most of the file system. Accordingly, the `rm` program will have an APO record that permits universal write access. Once lockdown is engaged and server programs are running so that customers can access them via the Internet, there is generally no use for the `rm` program. Accordingly, it must be made unavailable to attackers. The easiest way to accomplish this goal would be to remove all execution privileges from the `rm` program and then make the program immutable. Because lockdown prevents programs from being made mutable again, `rm` could not be executed during lockdown. Indeed, the suggestion that `rm` be made non-executable and immutable has been included in the `HS_lockdown.sh` script. After lockdown is disengaged, the `HS_unlock.sh` script can be used to return `rm` to its executable state so that it can be used during maintenance. Again, please note that if you do not run the `HS_unlock.sh` script before conducting maintenance, you may well encounter the error message explained in the section above.

Of course, certain programs require write access during lockdown. For example, a database server program stores database data in specific files. Consequently, the APO record for the database server must include write permission for the directory in which the database files are located, or at least write permission for those files. The APO record should not permit write access to other directories or files, though. As a result, attackers will not be able to manipulate the database server into writing to non-database files. Protection of database files, though, will ultimately rest with the database server program; that type of protection is beyond the reach of HeartSuite.

Sadly, there are occasions when a program that has universal write permissions must run during lockdown. For example, we found that a routine on Ubuntu 22.10 executes the `rm` program during shutdown. Accordingly, the program must remain available during lockdown to accomplish this task. Of course, if we left the actual `rm` program available, attackers could leverage it to possibly delete many essential files. Our solution was to create an exact copy of the `rm` program; we called it `limited_rm`. We created a script that is intended to be used only occasionally; it renames `rm` to `rm-orig`, then creates a symbolic link to `limited_rm` using the name `rm`:

```
# sudo cp /usr/bin/rm /usr/bin/limited_rm
# sudo mv /usr/bin/rm /usr/bin/rm-orig
# sudo ln -sf /usr/bin/limited_rm /usr/bin/rm
```

We then rebooted the system a few times to obtain all of the log entries needed for use of `limited_rm`. Once we added those specific directories to the APO record using the `add_start_and_shutdown_programs.py` script, we revised the `HS_lockdown.sh` script to remove execute privileges for `rm-orig`, then make both `rm-orig` and `limited_rm` immutable. Thus, during lockdown, any script calling the `rm` program-- whether during startup, shutdown, or intermittently by `systemd` or `cron`, will actually be invoking the `limited_rm` program, which has limited write access permissions. While this scenario opens certain, limited avenues for attack, it appears sufficient for overall server protection.

Notably, we also revised the `HS_unlock.sh` script to make both `rm-orig` and `limited_rm` mutable again and restore execute privileges to `rm-orig`. To return to full use for maintenance, we additionally created a counterpart to the "occasionally needed" script above; this script simply destroys the symbolic link by returning `rm-orig` to its original name:

```
# sudo mv /usr/bin/rm-orig /usr/bin/rm
```

After this latter script is run, subsequent scripts that invoke `rm` will be permitted universal write permissions, in accordance with `rm`'s APO record.

Protecting Your Server During Maintenance

We refer to maintenance as any time that you need to make changes to your server, such as adding or modifying packages or other files. As explained above, you will likely need to switch to Monitor mode to perform maintenance, let alone disengage lockdown. Because the `HS_startup.sh` script will return your server to lockdown every time you boot, you must boot to an alternate kernel in order to prevent it from being engaged.

Notably, the HeartSuite installation routine leaves your current kernel in place; it merely adds the HeartSuite-modified kernel and the HeartSuite tools. Thus, you can use the GRUB menu to select your prior kernel at bootup. Once you have booted to a non-HeartSuite kernel, you can use the `hs-monitor-state` program to switch back to Monitor mode. Alternatively, you can still work in Denial mode if you do not need to use tools that are incompatible with Denial mode, such as `dpkg`, but you will need to comment out the line in the `HS_startup.sh` script that calls the `HS_lockdown.sh` script.

Without Denial mode, and lockdown in particular, your server is once again vulnerable to attack! You must implement other security measures in order to reduce the chances of being attacked during maintenance. There are a few measures that we can readily suggest:

Blocks All Connection Requests

The simplest and likely most effective way to prevent connection requests is to disable interfaces to networks, using `ifdown` or a similar tool. You can add lines to either the `HS_startup.sh` script or the `HS_lockdown.sh` script so that these interfaces are enabled once HeartSuite has been reactivated in Denial mode but just prior to engaging lockdown. Of course, this protective measure requires that you have either physical or serial port access to your server in order to conduct maintenance.

For example, if the name of the interface that manages your server's network connection is eth0, then you can add the following lines to the startup script around the line that activates HeartSuite:

```
ifdown eth0
/.hs/sys/activate_HS
if [ $? -eq 0 ]; then
    ifup eth0
fi
```

The foregoing script does not re-enable the network interface if HeartSuite fails to activate. Alternatively, you can perform the same tasks using the `ip` command directly:

```
ip link set eth0 down
/.hs/sys/activate_HS
if [ $? -eq 0 ]; then
    ip link set eth0 up
fi
```

On the other hand, if you need to use SSH or other network-based tools to perform maintenance, you can set up firewall rules on your server that block all connection request traffic except for your maintenance connection. These firewall rules can be constructed to permit you to add and update packages using Aptitude (`apt`). You can add lines to the `HS_lockdown.sh` script that disables these rules after lockdown is successfully engaged. Thus, your server will be generally accessible from the Internet again but only after lockdown is engaged. Better yet, set up a separate firewall server between your server and the Internet, or even internal networks, that contains these firewall rules. You will have to manually disable the rules when you are ready for other computers to connect to your server again but the level of protection is potentially higher.

Shutdown Server Programs

Attackers often rely on server programs for access over the Internet. If your server programs are not running, that avenue of attack is foreclosed. Once you have completed your maintenance and rebooted with lockdown engaged, you can restart your server programs. The one exception is SSH, explained below.

Constrain SSH Access

Optimally, you can connect to your server using serial port access provided by a cloud provider. In that case, you can remove the script that starts `sshd` automatically after booting; instead, you must start it manually as needed. If you must use SSH for maintenance, however, you must constrain access to the extent possible. First, remove root login over SSH. Second, restrict SSH logins to only a single administrative user. Third, remove password access over SSH and use public/private key pairs instead. Fourth, implement a firewall rule that limits the IP addresses that can be used for SSH. Optimally, use a VPN for SSH access.

Changing the Cache size

HeartSuite maintains a cache of APO records during operation in order to improve performance. You may alter the size of the cache at the time of activation, in order to suite your needs, by using the `hs-APO-cache-size` program. In this version of HeartSuite, the minimum size of the cache is 10 APO records and the maximum size is 255 APO records. Each slot in the cache is reserved for a different program, which includes interpreted code files, as well. Thus, only a single APO record slot in the cache is needed to run multiple instances of the same program.

Appendix: List of HeartSuite tools included

With exception of the shim programs, all of the tools are located in the `/.hs/sys` directory. Please note that the HeartSuite installation routine does **NOT** add this directory to the `PATH` environment variable. The shim programs are located in the `/usr/bin` directory because it is in the default `PATH` variable. As mentioned above, the programs and scripts that write data to the HeartSuite databases must be run as root.

Executable Programs

A brief description of each program follows. The description also notes if a program includes help instructions, which may be accessed with the `--help` command line option or by starting the program without any command line arguments.

`activate_HS` -- turns HeartSuite service on; the installation routine adds a `systemd` service that automatically runs this program at startup

`hs-APO-cache-size` -- used to change the maximum number of APO records that may be cached simultaneously; please view program help for details

`hs-app-perm-orders-manager` – manages permission orders that allow programs to run, as well as which directories and remote computers they may access; please view program help for details

`hs-backup` -- used by HeartSuite to back up files; you never need to execute this program directly!

`hs-backup-config-manager` -- use this program to specify a list of directories; only files in directories designated using this program are backed up automatically by HeartSuite when modified

`hs-curfew` -- stops HeartSuite from backing up files; this state must be achieved so that HeartSuite does not interfere with normal shutdown; the setup procedure adds a `systemd` service that automatically executes this program just before a shutdown or reboot

`hs-monitor-state` -- used to change whether HeartSuite starts in Monitor or Denial mode; please view program help for details

`hs_python3, hs_python2, hs_perl, hs_php` -- shim programs included with this version `hs-unlock-progs` -- switches all HeartSuite files back to being mutable

`hs-shim-manager` -- configures the names of the interpreter programs that shim programs launch

`hs-verify-reg` – used by HeartSuite to verify registration; do not run this program directly

`hs-version-manager` -- permits restoring prior versions of backed up files; please view program help for details

`lockdown_HS` -- protects HeartSuite settings during Denial mode by making all HeartSuite files immutable

`register_HS_license` – registers the server using your HeartSuite license

Python scripts

Each script contains help information, which is displayed when the script is started without any command line arguments.

`add_start_and_shutdown_programs.py` – scans the HeartSuite log and the kernel log for permission error messages and uses them to construct appropriate APO records

`batch_record_add.py` -- adds programs listed in a file to APO records, as well as basic directory access

`batch_record_add_root.py` -- adds programs listed in a file to APO records, as well as read access to all files; USE THIS SCRIPT WITH CAUTION, IF AT ALL

`extract_program_names.py` -- extracts program names from a log file

`realpath_generator.py` -- generates the absolute path to programs

Shell scripts

These scripts do not include help information, but they are very simple to read and understand.

`empty_HS_log.sh` -- resets the HeartSuite log file to zero bytes

`init_base_records.sh` -- used by the installation script to add the Linux Standard Base (LSB) programs to APO records, as well as certain essential programs; this script is used only once--during Part 1 of installation

`HS_lockdown.sh` -- makes files and directories immutable using `chattr`, then executes the `lockdown_HS` program; EDIT THIS SCRIPT TO MEET YOUR NEEDS

`HS_startup.sh` – called by the `systemd` `heartsuite.service` unit immediately after booting; it activates HeartSuite automatically; by default, it also engages lockdown if Denial mode is chosen; EDIT THIS SCRIPT TO MEET YOUR NEEDS

`HS_unlock.sh` -- runs the `hs-unlock-progs` program and then makes other files and directories mutable again; this script is the counterpart to `HS_lockdown.sh`; EDIT THIS SCRIPT TO MEET YOUR NEEDS