##### HoneyBest User Manual 1.0.1

Jimmy Chen

[Jimmy.chen@moxa.com](mailto:Jimmy.chen@moxa.com)

2021/01/25

**HoneyBest user manual 1.0.1**

By Jimmy Chen

Copyright @ 2020-2021 Jimmy Chen

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1; with the Invariant Sections being "Introduction" and all sub-sections, with the Front-Cover Texts being "Original Author: Jimmy Chen", and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License". All scripts in this tutorial are covered by the GNU General Public License. The scripts are free source; you can redistribute them and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation, version 2 of the License. These scripts are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License within this tutorial, under the section entitled "GNU General Public License"; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

目錄

[How to read 4](#_Toc64638584)

[Prerequisites 5](#_Toc64638585)

[Conventions used in this document 6](#_Toc64638586)

[Chapter 1. Introduction 7](#_Toc64638587)

[Chapter 2. Prepare Your Environment 12](#_Toc64638588)

[Chapter 3. Understanding the System 13](#_Toc64638589)

[Chapter 4. Practical & Implementation 17](#_Toc64638590)

[Chapter 5. Conclusion 24](#_Toc64638591)

[Appendix A. 25](#_Toc64638592)

# How to read

This document was written to give everyone a good and simple primer at how to get started with HoneyBest security module, but at the same time it was created to be as complete as possible.

This document could either be read as a reference or from start to end. It aims at being an as complete reference as possibly to HoneyBest and to at least give a basic and fast primer or repetition to the areas that you might need to understand. There are some changing found in HoneyBest while it still in early stage of development, it’s inevitable. If you find peculiar bugs or behaviors in HoneyBest or any of the subcomponents, please sent me an email and tell me about the problem. If you have any suggestions on additions or if you think you find any problems around the area of HoneyBest not covered in this document feel free to contact me about this. I will be more than happy to take a look at it and possibly add what might be missing.

# Prerequisites

This document requires some previous knowledge about Linux/Unix, shell scripting, as well as how to compile your own kernel, and some simple knowledge about the kernel internals. I have tried as much as possible to eradicate all prerequisites needed before fully grasping this document, but to some extent it is simply impossible to not need some previous knowledge.

# Conventions used in this document

The following conventions are used in this document when it comes to commands, files and other specific information.

* Long code excerpts and command-outputs are printed like shown below. This includes screen dumps and larger examples taken from the console with green color.

*[root@work]# ls*

*default eth0 lo*

*[root@work]#*

* All commands and program names in the tutorial are shown in **bold typeface**. This includes all the commands that you might type, or part of the command that you type.
* All system items such as hardware, and also kernel internals or abstract system items such as the loopback interface are all shown in Courier typeface.
* Computer output is formatted in *this way* in the text. Computer output could be summed up as all the output that the computer will give you on the console.
* filenames and paths in the file-system are shown like ***/proc/honeybest/binprm.***
* ME refer to “Management Entry”.
* DE refer to “Data Entry”.

# Chapter 1. Introduction

##### Why I create new LSM

Over the years few security modules have been developed on Linux distribution, such as SELinux / Apparmor / Smack / Tomoyo project, but there is still huge space to make improvement nevertheless. Until now, the high entry barrier keep apart from most of the Linux user. For those who have little understanding of system behavior & security thread model are frustrate to apply the software. In order to build the more user friendly module, our target is to hide the complexity of rules, but also allow advanced user to be able to refine the granularity.

Most of the case, security module begin to involve during post software development stage. Once software product deliver to security expert, he has to stand on top of system view to be able to write rules to protect applications, configuration files and resources. In order to do so, he has to go deep understanding through what exactly every single process do to prevent from threats. However, in reality world, neither security expert get involve into software development cycle nor have software developer background. This brings up the question, is there any possible we can build a better solution based on auto generation policy base? How if we could invent a new secure module policy to support interaction with developer whether or not to add new rules or requesting permission under safe condition? Is there an alternative approach to replace rules concept? Well, HoneyBest secure module might be for those answer.

##### Terms used in this document

This document contains a few terms that may need more detailed explanations before you read them. This section will try to cover the most obvious ones and how I have chosen to use them within this document.

Vector - This is generally referred to in this document as a kernel activity features. In normal circumstances, kernel activities are combination of user space process’s request such as file open, file execute, socket control, change permission and so on. Vector tuning allow us to selectively filter kernel activity hence auto generate policies and notification.

Interactive – Interactive referred to the mode use to collect kernel activities into single proc file, depended by Vector.

##### Concept & Design

Let us imaging few conditions here.

Condition A – Environment complexity is hard to apply rules

Team of developers have complete their software development on Linux box. The appliance involve NGINX server for user to configure setting; Samba server for file sharing; SNMP server for remote setting; Syslog server to track system record. They handle the appliance to one of their security guy, Bob, who are the expertise in security module. In order to create threat model, Bob have to understand every single process running on the box, how each process interfere with system and other processes. He now create rules to protect base on the threat model. At first, he create rules to restrict process to access only certain system resource, such as Syslog server. Syslog server is allowed to create files under /var/log/\*.log, with WRITE permission only; Syslog server is allow to create only localhost 514 UDP port, receive other application log message. Here come small part of complicate scenario, log message files could grow up over the time and Logrotate daemon are design into system to handle compression job; log message files need permission rules to MOVE files(DELETE/CREATE/READ/WRITE); Meanwhile, NGINX we server need permission READ in order to show context while user login via web page. After Bob figure out all those cross over relationship & rules, he start to apply into system. It turn out, the box does not act as normal as expect to pass system integration test. Bob have to invite developer to figure out what going on to the system. It turn out that NGINX web server need permission rules to interact with 514 UDP port for logging itself message.

In real world, security expertise feel frustrate to do their job because of complexity environment involve.

Condition B – High learning curves

User, roles, level, category, labeling, and hats are not easy to understand, those are security expertise concept with specific tools. Most of the small/medium company do not have security expertise to rely on. We want to help software developers secure their product as much as we can.

Condition C – Untrusted root in design

he complete security policies should treat super user (root) as normal root. Root are not allow to change others policies but its own. Penetration to root user might corrupt whole policies wall you made. In our design, policies update or change should bind tightly with secure boot process, more precisely, with hardware Root of Trust.

Condition D – Interaction in real time instead of post rules applied

Real time interaction feedback mechanism are more easy way for developers to understand what going. Instead of rules, pop out dialogue asking permission to explain activity is an effective way to make progress. For the fine-grain advanced user, our design also consider to fulfill such needs.

Condition E – Different perspective of software protection

In some privacy scenario, system designer not only require the task to have restriction from accessing resources, but also restriction from other resources to access the task. Here are the 2 examples, I want to protect my private libraries/program from piracy, however, still allow certain program to use; I want only “upgrade-firmware” command to be able upgrade system firmware, not “dd” command, and the integrity of “upgrade-firmware” command is concerned.

The core design of HoneyBest is to focus on capturing the kernel activities triggered by user space program. Activities which is tracking (Vector) will later turn into a list data structure for security module to detect an unexpected occur event. The size of list data structure is tightly depends on level of granularity and vector spectrum. The more precise kernel activities to be captured, the higher space requirement for data structure to be saved. Above the surface of such design, here is the approach to apply secure module. Prepare your security environment, run activities as you normal condition to create a new model, then use these model as your policy rules. Once the model is applied, all activities are under restrict to the model. You might consider fine-grain the model because some activities are not able to perform in your security environment. Either use an editor to edit the model or turn on interaction mode, you are able to selectively choose exception with new activity in real world situation. Below figure show how the lifecycle go:

Product finish development

1st step, turn on capture mode and turn off interactive mode, start end to end System Integration Test.

2nd step, turn on interactive mode, start end to end System Integration Test. If an exception find in interactive mode proc file, you can manually edit the model according to interactive output or copy as your reference.

Turn off interaction mode.

3rd step, turn off all mode. Copy all vector model and apply to your production.

# Chapter 2. Prepare Your Environment

##### Compiling

Similar to SELinux/Apparmor design, HoneyBest security module is hooked on Linux Security Module layer. Clone the source code into Linux kernel source and follow instruction below:

1. Create a new directory called HoneyBest under [KERNEL SOURCE]/security directory.
2. Clone HoneyBest source code into a new directory.
3. If you are Debian/Ubuntu environment, install necessary packages to compile new kernel (apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev bc).
4. Change directory to [KERNEL SOURCE] and run the command #patch –p1 < security/honeybest/patches/kernel-4.19.patch
5. Copy original kernel configuration to [KERNEL SOURCE]/.config (cat /boot/config-4.9.X > [KERNEL SOURCE/.config.
6. Select HoneyBest security module (make menuconfig)
7. Compiling kernel under [KERNEL SOURCE] (make modules bzImage)
8. Install new kernel & modules (make install)

##### Note

Current HoneyBest source code has been compatibility test with Linux kernel version 4.19.0-14 both armhf and amd64.

# Chapter 3. Understanding the System

##### General

Before we go through this chapter, keep in mind that HoneyBest is designed to bind with Linux secure boot process. In order to do so, it needs to be function in limit resources environment such as Initramdisk or InitramFS. Hence, you are able to setup HoneyBest only if below environment and utilities (busybox):

* Command utilities
  + **cat**
  + **echo**
  + **bash** or **dash**
* Environment
  + Dash / Bash redirect features
  + Linux PROC file system

The overall HoneyBest features are setup via PROC files, either enable/disable certain set of features (Vector) or policy rules.

Example below enable HoneyBest:

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/enabled*

*[root@work]# cat /proc/sys/kernel/honeybest/enabled*

*1*

There is 2 main design in HoneyBest internal, management and data entries. Management entries is use to control HoneyBest behavior and Data entries use to read and write policies. Their working directory are separately locate under PROC file system.

Management entries – Directory */proc/sys/kernel/honeybest*

Data entries – Directory */proc/honeybest*

##### Management entry (ME)

Management entry are the step to process during system boot up. First, let us look at the files show under this directory and their corresponding functions.

Table 1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File | Default value | Options | Description | Note |
| enabled | 0 - disable | 0 / 1 | Enable/Disable option. It needed to combine with Vector option. |  |
| interact | 0 - disable | 0 / 1 | This option use to separate all |  |
| audit | 0 - disable | 0 / 1 | Enable/Disable auditing with auditd. |  |
| locking | 0 - disable | 0 / 1 | Once you turn on locking, system will prohibit any read/write operation to Data entry, except when interact value is 1. For security reason, this features is useful to lock down anonymous who try to modify after initramfs chroot to /. |  |
| bl | 0 - whitelist | 0 / 1 | Whitelist / Blacklist as default policy. |  |
| level | 1 - granularity | 1 / 2 / 3 | The higher granularity, the more restrict policies will auto generate into your Data entry. |  |
| binprm | 0 - disable | 0 / 1 | Enable tracking executable file | Vector |
| files | 0 - disable | 0 / 1 | Enable tracking ordinary file | Vector |
| inode | 0 - disable | 0 / 1 | Enable tracking inode operation | Vector |
| ipc | 0 - disable | 0 / 1 | Enable tracking Linux internal process communication | Vector |
| kmod | 0 - disable | 0 / 1 | Enable tracking Linux kernel modules | Vector |
| path | 0 - disable | 0 / 1 | Enable tracking device node, hard/soft symbolic, directory, pipe, unix socket | Vector |
| ptrace | 0 - disable | 0 / 1 | Enable tracking ptrace activities | Vector |
| sb | 0 - disable | 0 / 1 | Enable tracking superblock information | Vector |
| socket | 0 - disable | 0 / 1 | Enable tracking TCP/UDP/ICMP socket | Vector |
| task | 0 - disable | 0 / 1 | Enable tracking activity between process | Vector |

##### Data entry (DE)

Every single files in directory /proc/honeybest tracking different behavior. We will explain each single file corresponding on next section. In general, every file share the common column, e.g NO/FUNCTION/USER ID.

* NO – sequence number, honeybest compare the occurrence activities begin from lower to higher number.
* FUNCTION – functional identification, honeybest use to identify different activities. Under certain category such as ‘socket’, different activities are label as listen/bind/accept/open/setsocketopt and so on.
* USER ID – user identification, honeybest use to reference relationship between identity and function. This column support RE(regular expression, digits & '\*' asterisk).
* ACTION - Matching action refer to 'A'ccept or 'R'eject. Default value depend on bl option, accept actions are appended when bl toggle to 0; vice versa, reject actions are appended.

Table 2.0

|  |  |  |
| --- | --- | --- |
| File | Description | Comment |
| binprm | Tracking all executable file path name, process UID belong to and most importantly, calculate file context into HASH to protect the integrity. |  |
| files | Tracking ordinary file behavior, such as open /read /write /delete /rename. |  |
| inode | Tracking inode operation, such as create /delete /read /update /setxattr /getxattr |  |
| path | Tracking behavior of all type of file such as device node, hard/soft symbolic, directory, pipe, unix socket. |  |
| socket | Tracking TCP/UDP/ICMP socket activity, including port number. |  |
| task | Tracking activity between process, such as signal exchanging. |  |
| sb | Tracking superblock information. Activities such as mount/umount/df will stamp into this category. Highly relate to file/path categories due to system register /proc information. |  |
| kmod | Tracking Linux kernel modules activity. Kernel modprobe will stamp into this category. |  |
| ptrace | Tracking ptrace activities. |  |
| ipc | Tracking Linux internal process communication activities such as share memory, message queue & semaphore. |  |
| notify | Notification between security module and user space application. In interactive mode, detect to unexpected events are save into this file for user space program to notify user later. Dialogue pop up to acquiring security expertise allow or ignore such activities. Once the interactive mode is enable, all events go through this file could expose memory exhaust. Thurs, design a READ scheduler from user space program is vital. Context in notify file will be cleaned after each single READ operation is executed. |  |

# Chapter 4. Practical & Implementation

As we have already explained to some concepts and features, this chapter will describe how we use HoneyBest in our Linux operating system. I will give you a good and simple primer at how to get start with examples. The tutorial will expose in few scenarios and cover what exactly to do.

##### Protect your running environments

The simplest task we want to do with HoneyBest is protecting our software environment. Section here point out how easily to use HoneyBest if you have a product built up with Linux environment. To make it simple how it work, let start our goal to protect the normal user who can run binary command that we approved. Here, keep in mind the binary command we like to protect involves 2 major elements, integrity of binary itself and the location.

First, let us login 2 terminal consoles with super user (root) and (bob) user. We need root user to setup HoneyBest, whist bob user use to verifying the result root user set. Once you have both user login into the console, now we can start to enable HoneyBest with below command:

(root user)

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/enabled*

*[root@work]# cat /proc/sys/kernel/honeybest/enabled*

*1*

Remember the Table 1.0 we had already mentions in chapter 3 [ME]? File “binprm” locate under directory /proc/sys/kernel/honeybest is corresponding to tracking executable binary. Let give it a try and see what happen after we enable the option.

(root user)

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/binprm*

*[root@work]# cat /proc/sys/kernel/honeybest/binprm*

*1*

If you recall what we describe [DE] in chapter 3, all execute binary digest hash and path will be recorded immediately after we enable the HoneyBest. Check out below:

(root user)

*[root@work]# head –n 3 /proc/honeybest/binprm*

*NO FUNC UID ACTION DIGEST PATH*

*0 2000 0 A 67fd5fdf3161a0c086932074844a8bbf444b8911 /bin/bash*

*1 5003 0 A 709ee3af19ba791b699fea9127df0da6dc2bc248 /lib/arm-linux-gnueabihf/ld-2.24.so*

Let me explain the command output above. The first line show title of output, which include number of line [NO]; function captured by kernel [FUNC]; process owner [UID]; rules acceptance [ACTION]; hash value [DIGEST] and binary absolutely path [PATH]. The explanation to second line indicate that once we enable binprm tracking, kernel capture there is super user who execute the bash command. The third line show the shared library (ld-2.24.so) is called by bash binary during execute process.

There is more to tell you about the information above, but let us start with [ACTION] title. The default to control the global entry as a whitelist is [bl] vector, which we already mentioned in [ME] Chapter 3. That means, once we turn locking mode into enable, any binary path that list on binprm file will be examined. If the binary path match the list, then [ACTION] will be later examined. “A” in [ACTION] title indicate accept, whist “R” in [ACTION] title indicate reject. In our case here, bash & ld-2.24.so are allow to execute.

In the opposite, we can now try to examine what happen if no rule match to our binprm list. Let us first clean the binprm list with super user here.

(root user)

*[root@work]# echo “” > /proc/honeybest/binprm*

Now, we enable the locking option so that binprm will be examined.

(root user)

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/locking*

Let try to run “date” command on bob user console terminal.

(normal user)

*[bob@work]# date*

*-bash: /bin/date: Operation not supported*

As you can expect, ‘date’ command fail to execute because the rule does not appear in binprm list. Our final exam will be add rule into binprm list and test again the result. First, we need to disable the locking option.

(root user)

*[root@work]# echo 0 > /proc/sys/kernel/honeybest/locking*

Now, we re-run the ‘date’ command under normal user console, so that date will be automatically add to binprm list.

(normal user)

*[bob@work]# date*

*Fri Jan 29 06:38:29 UTC 2021*

Enable again the locking option under super user (root).

(root user)

*[root@work]# echo 0 > /proc/sys/kernel/honeybest/locking*

You might discover the ‘date’ command now can be executed under normal user (bob) even we turn on the locking option.

(normal user)

*[bob@work]# date*

*Fri Jan 29 06:38:29 UTC 2021*

In order to verify the rule, run ‘grep’ under super user console. See below:

(root user)

*[root@work]# cat /proc/honeybest/binprm |grep date*

*34 2000 0 A 0dd0e8feb70879329e11bff7aedf37e81a236aff /bin/date*

*63 2000 1000 A 0dd0e8feb70879329e11bff7aedf37e81a236aff /bin/date*

As you can see, the second line (UID = 1000) had been added into binprm list once locking option = 0, after bob user run ‘date’ command.

Let make a quick overall what we have learnt on above examine. HoneyBest default locking option = 0 will auto add kernel activities into corresponding vector’s file, which in our case here, execute binary tracking (binprm file). Before all this happen, enabled and binprm vector’s files should fill with value 1. Our next exam will go to more deep to how to tuning the rules.

Let us recall to our previous command.

(root user)

*[root@work]# head –n 3 /proc/honeybest/binprm*

*NO FUNC UID ACTION DIGEST PATH*

*0 2000 0 A 67fd5fdf3161a0c086932074844a8bbf444b8911 /bin/bash*

*1 5003 0 A 709ee3af19ba791b699fea9127df0da6dc2bc248 /lib/arm-linux-gnueabihf/ld-2.24.so*

As we mention in Chapter 3 [DE] section, [ACTION] column refer to single rule should be accepted or rejected. HoneyBest will scan the rules start from [NO] 0 to the end of rows. Let change normal user ‘date’ rule definition in binprm file and see what will happen. We need to save our current policy rules into file, editing the [ACTION] column, then apply back to HoneyBest. Fortunately, our KISS design in HoneyBest make life easier than you thought it will be.

(root user)

*[root@work]# echo 0 > /proc/sys/kernel/honeybest/binprm*

*[root@work]# echo 0 > /proc/sys/kernel/honeybest/locking*

*[root@work]# cat /proc/honeybest/binprm > /root/binprm*

*[root@work]# vi /root/binprm*

Once we save binprm to /root directory, use any editor such as Vim to find the date string and replace the ‘A’ at column [ACTION] to ‘R’ (Remember to replace the rule at user who match the [UID], in our example here bob uid is 1000).

One important thing to reminding before your last step to save the file. The [DE] files default attach 1st row with description and 1st column numeric for readable purpose. Hence, you need to remove 1st row and column to make it into HoneyBest understandable data. With Vim editor, move your cursor to 1st row and press {dd} key; move your cursor to 1st row, use {ctrl+v}, {G}, move cursor to the end of 1st column and then {d} to wipe out. Detail vim can be referenced to here <https://askubuntu.com/questions/531271/how-to-delete-second-column-in-vim> .

(root user)

*[root@work]# cat /root/binprm > /proc/honeybest/binprm*

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/binprm*

*[root@work]# echo 1 > /proc/sys/kernel/honeybest/locking*

As we expect again, the command ‘date’ under user console fail to execute even binprm list contain the rule.

(normal user)

*[bob@work]# date*

*-bash: /bin/date: Operation not supported*

What we have learnt so far is [ACTION] have higher priority than default ‘bl’ option. If you are familiar with iptables, [bl] option have someway similarity to default chain policy while [ACTION] equal to “-j” options.

Well, that all in this section. In HoneyBest 1.0, 10 vectors are provide for you to tracking your kernel activities. [ME] act as control system whist [DE] act as setting. [enabled] options in [ME] use to enable the HoneyBest. Selectively enable which vector you want by written value “1” into [ME] vector files. Default value 0 in locking option will automatically add kernel activities into [DE] vector files. Write value 1 into locking option will turn tracking mode to defending mode. In defending mode, rules and [ACTION] are referenced according to priority.

##### Protect your critical shared libraries

In this section, we will bring you into HoneyBest advances topic. In order to systematically help you understand what going on, again, I will instruct you with an example in this section.

Let say our objective is to protect TPM shared libraries, whist, provide a Linux environment for root user and normal user to operate. Trust Platform Module is a safety module hardware attach on the machine. We make an assumption that the libraries can be penetrated by hacker(or root user) who have high knowledgeable understanding of internal process have capability to replace it with vulnerable libraries. In our example here, we want to protect these shared libraries list below from scp/copy/move out of box, in the same time, we also would like to provide utility (tpm2-nvlist) to use these shared libraries during execution.

Below is the TPM shared libraries files location:

* /usr/lib/arm-linux-gnueabihf/libtss2-sys.so.0.0.0
* /usr/lib/arm-linux-gnueabihf/libtss2-esys.so.0.0.0
* /usr/lib/arm-linux-gnueabihf/libtss2-mu.so.0.0.0
* /usr/lib/arm-linux-gnueabihf/libtss2-tcti-mssim.so.0.0.0
* /usr/lib/arm-linux-gnueabihf/libcrypto.so.1.1
* /usr/lib/arm-linux-gnueabihf/libtss2-tcti-device.so.0.0.0

Below is the utility files location:

* /usr/bin/tpm2\_nvlist

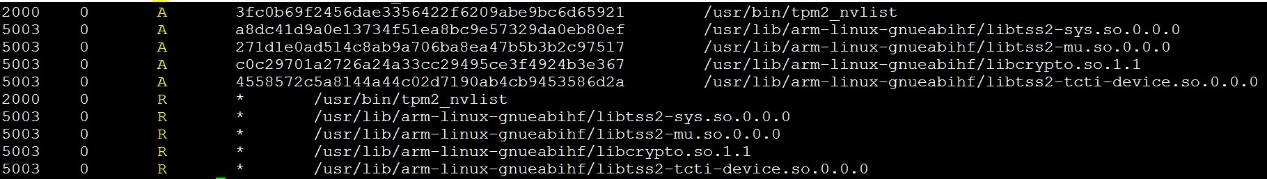
You need to enabling/design secure boot process in order to prohibit kernel & initramfs from replacing. In addition, we suggesting use hardware security module (HSM) such as TPM/ArmTrustZone to involve into secure boot process. Reformat your partition with LUKs and bind LUKs's key to HSM. Here are the few steps:

Recompiling kernel option with CONFIG\_HONEYBEST\_PROD=y. Create a new directory under /etc/honeybest.

1. Add 'files' feature set configuration into initramfs, save it into directory /etc/honeybest/files:



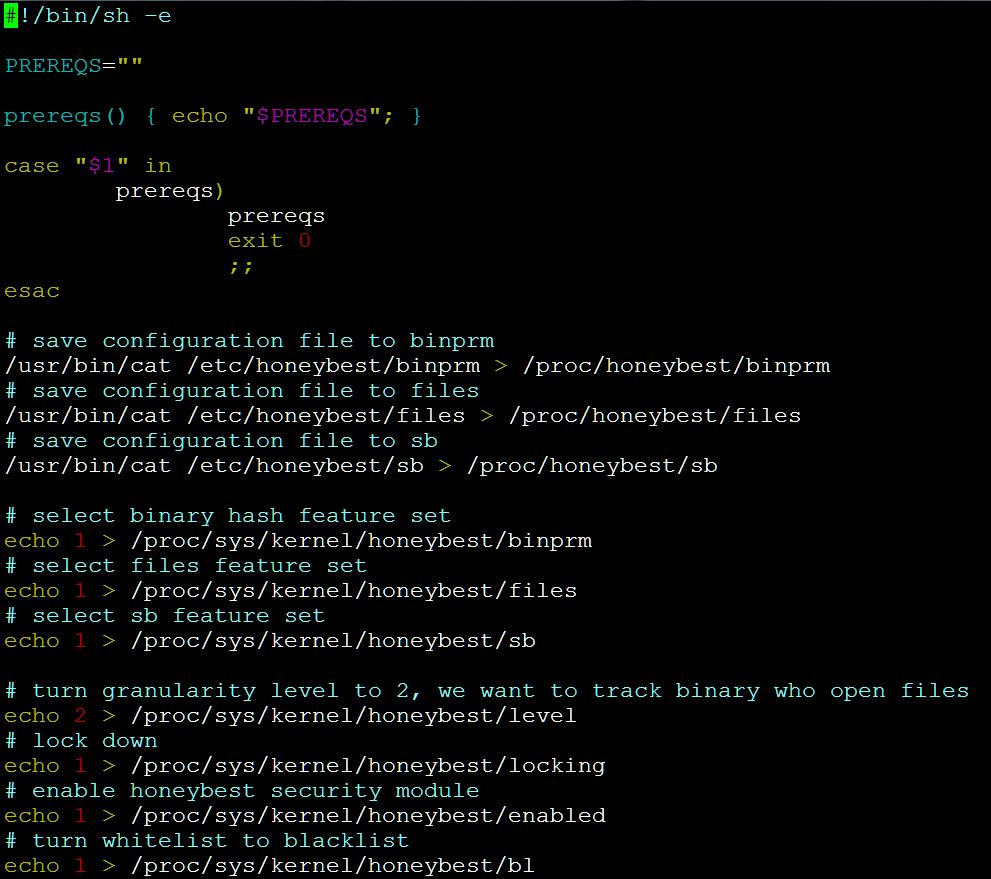
1. Add 'binprm' feature set configuration into initramfs, save it into directory /etc/honeybest/binprm:



1. Add 'sb' feature set configuration into initramfs, save it into directory /etc/honeybest/sb



1. Add initramfs script (init-top) to run before chroot into LUKs filesystem:



It looks weird for the first time when you see the rules. However, if you start to think of that we want to build a normal Linux environment but at the same time still restrict user to access certain libraries and utility, it does make sense after all.

Let see the last figure (step 4) before we begin to describe the rules here. As you can see, the last command from last figure turn default policy from whitelist to blacklist. That mean, any rules that do not match will be allowed to execute by default. Later inside the different [DE], we manipulate the [ACTION] to make it allow for certain libraries and utility, then reject all to access libraries. The shared libraries and utility had been hashed in [DIGEST] binprm file. Any change to binary will not allow to execute hence protect the binary from replacement or penetrated.

##### Event monitoring & appending new rule

It is important to monitor your operating system environment while system is running in order to identify potential risk. HoneyBest is able to help you capture the task activities from kernel level perspective. Before we go further, here are few variable matric in [ME] that are involves in controlling the output.

Table 2.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vector |  | locking | Description | Note |
| interact | 0 | 1 | HoneyBest start to match the rules in [DE]. | The default policy is depend on bl vector. |
| interact | 1 | 0 | HoneyBest compare rules in [DE], exception rules will redirect to [DE] notify file. | The redirect rules list is depend on bl vector. |
| interact | 1 | 1 | HoneyBest start to match/capture the rules in [DE]. Incompatible match rules will attach to [DE] notify file. | The redirect rules list is depend on bl vector. |
| interact | 0 | 0 | HoneyBest start to capture the rules found in [DE]. Incompatible match rules will attach to [DE] files itself. | The redirect rules list is depend on bl vector. |

Let say if you already have certain rules in [DE], and the locking is turn on to protect your environment. However, at the same time, you also want any activities that might causes an exception or notifying to your remote system. Turn on interact vector is the most suitable in this case. All the activities that fail to match the [DE] will redirect to notify file (/proc/honeybest/notify). The administrator is able to add new rules into [DE] by referenced the notify file content. It is important to notice that notify file are designed as a temporary queueing file, the content immediately swap out after read operation have done. Keep in mind that you need to constantly read the file to prevent memory exhaust.

# Chapter 5. Conclusion

HoneyBest project is not the replacement of SELinux / Apparmor and other security modules. It is another selective option for Linux administrator who like to protect their Linux product in fast and easy way. Any contribution or feedback are most welcome to make it to become a better project. Also, if case you’re willing to read more about HoneyBest, Appendix A is good resource to start.

# Appendix A.

HoneyBest Project URL

<https://github.com/u7702045/honeybest>