Bastille Documentation

Release 0.7.20200714-beta

Christer Edwards

Jul 14, 2020

Contents:

Welcome to the official Bastille documentation. This collection of documents will outline installation and usage of Bastille.

The latest version of this documentation can always be found at https://docs.bastillebsd.org.

CHAPTER 1

Installation

Bastille is available in the official FreeBSD ports tree at *sysutils/bastille*. Binary packages available in *quarterly* and *latest* repositories.

Current version is *0.7.20200714*.

To install from the FreeBSD package repository:

- quarterly repository may be older version
- latest repository will match recent ports

1.1 PKG

```
pkg install bastille
```

To install from source (don't worry, no compiling):

1.2 ports

```
make -C /usr/ports/sysutils/bastille install clean
```

1.3 GIT

```
git clone https://github.com/BastilleBSD/bastille.git
cd bastille
make install
```

This method will install the latest files from GitHub directly onto your system. It is verbose about the files it installs (for later removal), and also has a *make uninstall* target.

CHAPTER 2

Network Requirements

Here's the scenario. You've installed Bastille at home or in the cloud and want to get started putting applications in secure little containers, but how do I get these containers on the network?

Bastille tries to be flexible about how to network containerized applications. Three methods are described here. Consider each options when deciding which design work best for your needs. One of the methods works better in the cloud while the others are simpler if used in local area networks.

Note: if you are running in the cloud and only have a single public IP you may want the Public Network option. See below.

CHAPTER 3

Local Area Network

I will cover the local area network (LAN) method first. This method is simpler to get going and works well in a home network (or similar) where adding alias IP addresses is no problem.

3.1 Shared Interface (IP alias)

In FreeBSD network interfaces have different names, but look something like *em0*, *bge0*, *re0*, etc. On a virtual machine it may be *vtnet0*. You get the idea. . .

Bastille allows you to define the interface you want the IP attached to when you create it. An example:

```
bastille create alcatraz 12.1-RELEASE 192.168.1.50 em0
```

When the *alcatraz* container is started it will add *192.168.1.50* as an IP alias to the *em0* interface. It will then simply be another member of the hosts network. Other networked systems (firewall permitting) should be able to reach services at that address.

This method is the simplest. All you need to know is the name of your network interface and a free IP on your current network.

Bastille tries to verify that the interface name you provide it is a valid interface. It also checks for a valid syntax IP4 or IP6 address.

3.2 Virtual Network (VNET)

(Added in 0.6.x) VNET is supported on FreeBSD 12+ only.

Virtual Network (VNET) creates a private network interface for a container. This includes a unique hardware address. This is required for VPN, DHCP, and similar containers.

To create a VNET based container use the *-V* option, an IP/netmask and external interface.

```
bastille create -V azkaban 12.1-RELEASE 192.168.1.50/24 em0
```

Bastille will automagically create the bridge interface and connect / disconnect containers as they are started and stopped. A new interface will be created on the host matching the pattern *interface0bridge*. In the example here, *em0bridge*.

The *em0* interface will be attached to the bridge along with the unique container interfaces as they are started and stopped. These interface names match the pattern *eXb_bastilleX*. Internally to the containers these interfaces are presented as *vnet0*.

VNET also requires a custom devfs ruleset. Create the file as needed on the host system:

```
## /etc/devfs.rules (NOT .conf)

[bastille_vnet=13]
add include $devfsrules_hide_all
add include $devfsrules_unhide_basic
add include $devfsrules_unhide_login
add include $devfsrules_jail
add path 'bpf*' unhide
```

Lastly, you may want to consider these three *sysctl* values:

```
net.link.bridge.pfil_bridge=0
net.link.bridge.pfil_onlyip=0
net.link.bridge.pfil_member=0
```

CHAPTER 4

Public Network

In this section I'll describe how to network containers in a public network such as a cloud hosting provider (AWS, digital ocean, vultr, etc)

In the public cloud you don't often have access to multiple private IP addresses for your virtual machines. This means if you want to create multiple containers and assign them all IP addresses, you'll need to create a new network.

4.1 loopback (bastille0)

What I recommend is creating a cloned loopback interface (*bastille0*) and assigning all the containers private (rfc1918) addresses on that interface. The setup I develop on and use Bastille day-to-day uses the *10.0.0.0/8* address range. I have the ability to use whatever address I want within that range because I've created my own private network. The host system then acts as the firewall, permitting and denying traffic as needed.

I find this setup the most flexible across all types of networks. It can be used in public and private networks just the same and it allows me to keep containers off the network until I allow access.

Having said all that here are instructions I used to configure the network with a private loopback interface and system firewall. The system firewall NATs traffic out of containers and can selectively redirect traffic into containers based on connection ports (ie; 80, 443, etc.)

First, create the loopback interface:

```
ishmael ~ # sysrc cloned_interfaces+=lo1
ishmael ~ # sysrc ifconfig_lo1_name="bastille0"
ishmael ~ # service netif cloneup
```

Second, enable the firewall:

```
ishmael ~ # sysrc pf_enable="YES"
```

Create the firewall rules:

4.2 /etc/pf.conf

```
ext_if="vtnet0"

set block-policy return
scrub in on $ext_if all fragment reassemble
set skip on lo

table <jails> persist
nat on $ext_if from <jails> to any -> ($ext_if)

## static rdr example
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45

## dynamic rdr anchor (see below)
rdr-anchor "rdr/*"

block in all
pass out quick modulate state
antispoof for $ext_if inet
pass in inet proto tcp from any to any port ssh flags S/SA modulate state

# If you are using dynamic rdr also need to ensure that the external port
# range you are using is open
# pass in inet proto tcp any to any port <rdr-start>:<rdr-end>
```

- Make sure to change the *ext_if* variable to match your host system interface.
- Make sure to include the last line (*port ssh*) or you'll end up locked out.

Note: if you have an existing firewall, the key lines for in/out traffic to containers are:

```
nat on $ext_if from <jails> to any -> ($ext_if)

## static rdr example
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45
```

The *nat* routes traffic from the loopback interface to the external interface for outbound access.

The *rdr pass …* will redirect traffic from the host firewall on port X to the ip of Container Y. The example shown redirects web traffic (80 & 443) to the containers at *10.17.89.45*.

> ## dynamic rdr anchor (see below) rdr-anchor "rdr/*"

The *rdr-anchor "rdr/*"* enables dynamic rdr rules to be setup using the *bastille rdr* command at runtime - eg.

> bastille rdr <jail> tcp 2001 22 # Redirects tcp port 2001 on host to 22 on jail bastille rdr <jail> udp 2053 53 # Same for udp bastille rdr <jail> list # List dynamic rdr rules bastille rdr <jail> clear # Clear dynamic rdr rules
>
> Note that if you are redirecting ports where the host is also listening (eg. ssh) you should make sure that the host service is not listening on the cloned interface - eg. for ssh set sshd_flags in rc.conf
>
> sshd_flags="-o ListenAddress=<hostname>"

Finally, start up the firewall:

```
ishmael ~ # service pf restart
```

At this point you'll likely be disconnected from the host. Reconnect the ssh session and continue.

This step only needs to be done once in order to prepare the host.

CHAPTER 5

Usage

```
ishmael ~ # bastille help
Bastille is an open-source system for automating deployment and management of
containerized applications on FreeBSD.

Usage:
  bastille command TARGET [args]

Available Commands:
  bootstrap   Bootstrap a FreeBSD release for container base.
  cmd         Execute arbitrary command on targeted container(s).
  clone       Clone an existing container.
  console     Console into a running container.
  convert     Convert a Thin container into a Thick container.
  cp          cp(1) files from host to targeted container(s).
  create      Create a new thin container or a thick container if -T|--thick option␣
↪specified.
  destroy     Destroy a stopped container or a FreeBSD release.
  edit        Edit container configuration files (advanced).
  export      Exports a specified container.
  help        Help about any command.
  htop        Interactive process viewer (requires htop).
  import      Import a specified container.
  list        List containers (running and stopped).
  mount       Mount a volume inside the targeted container(s).
  pkg         Manipulate binary packages within targeted container(s). See pkg(8).
  rdr         Redirect host port to container port.
  rename      Rename a container.
  restart     Restart a running container.
  service     Manage services within targeted container(s).
  start       Start a stopped container.
  stop        Stop a running container.
  sysrc       Safely edit rc files within targeted container(s).
  template    Apply file templates to targeted container(s).
  top         Display and update information about the top(1) cpu processes.
```

(continues on next page)

(continued from previous page)

```
  umount      Unmount a volume from within the targeted container(s).
  update      Update container base -pX release.
  upgrade     Upgrade container release to X.Y-RELEASE.
  verify      Compare release against a "known good" index.
  zfs         Manage (get|set) zfs attributes on targeted container(s).

Use "bastille -v|--version" for version information.
Use "bastille command -h|--help" for more information about a command.
```

CHAPTER 6

Targeting

Bastille uses a *command target arguments* syntax, meaning that each command requires a target. Targets are usually containers, but can also be releases.

Targeting a container is done by providing the exact containers name.

Targeting a release is done by providing the release name. (Note: do not include the *-pX* point-release version.)

Bastille includes a pre-defined keyword ALL to target all running containers.

In the future I would like to support more options, including globbing, lists and regular-expressions.

CHAPTER 7

Examples: Containers

```
ishmael ~ # bastille ...
```

<table>
<tr><th>command</th><th>target</th><th>args</th><th>description</th></tr>
<tr><td>cmd</td><td>ALL</td><td>‘sockstat -4’</td><td>execute <i>sockstat -4</i> in ALL containers (ip4 sockets)</td></tr>
<tr><td>console</td><td>mariadb02</td><td>—</td><td>console (shell) access to mariadb02</td></tr>
<tr><td>pkg</td><td colspan="2">web01 | ‘install nginx’</td><td>install nginx package in web01 container</td></tr>
<tr><td>pkg</td><td>ALL</td><td>upgrade</td><td>upgrade packages in ALL containers</td></tr>
<tr><td>pkg</td><td>ALL</td><td>audit</td><td>(CVE) audit packages in ALL containers</td></tr>
<tr><td>sysrc</td><td>web01</td><td>nginx_enable=YES</td><td>execute <i>sysrc nginx_enable=YES</i> in web01 container</td></tr>
<tr><td>template</td><td>ALL</td><td>username/base</td><td>apply <i>username/base</i> template to ALL containers</td></tr>
<tr><td>start</td><td>web02</td><td>—</td><td>start web02 container</td></tr>
<tr><td colspan="4">cp | bastion03 | /tmp/resolv.conf-cf etc/resolv.conf | copy host-path to container-path in bastion03</td></tr>
<tr><td>create</td><td>folsom</td><td>12.1-RELEASE 10.17.89.10</td><td>create 12.1 container named <i>folsom</i> with IP</td></tr>
</table>

CHAPTER 8

Examples: Releases

```
ishmael ~ # bastille ...
```

command	target	args	description
bootstrap	12.1-RELEASE	—	bootstrap 12.1-RELEASE release
update	11.4-RELEASE	—	update 11.4-RELEASE release
upgrade	11.3-RELEASE	11.4-RELEASE	update 11.4-RELEASE release
verify	11.4-RELEASE	—	update 11.4-RELEASE release

CHAPTER 9

Bastille sub-commands

9.1 bootstrap

The bootstrap sub-command is used to download and extract releases and templates for use with Bastille containers. A valid release is needed before containers can be created. Templates are optional but are managed in the same manner.

Note: your mileage may vary with unsupported releases and releases newer than the host system likely will NOT work at all. Bastille tries to filter for valid release names. If you find it will not bootstrap a valid release, please let us know.

In this document we will describe using the *bootstrap* sub-command with both releases and templates. We begin with releases.

9.1.1 Releases

Example

To *bootstrap* a release, run the bootstrap sub-command with the release version as the argument.

```
ishmael ~ # bastille bootstrap 11.4-RELEASE [update]
ishmael ~ # bastille bootstrap 12.1-RELEASE
```

This command will ensure the required directory structures are in place and download the requested release. For each requested release, *bootstrap* will download the base.txz. These files are verified (sha256 via MANIFEST file) before they are extracted for use.

Tips

The *bootstrap* sub-command can now take (0.5.20191125+) an optional second argument of "update". If this argument is used, *bastille update* will be run immediately after the bootstrap, effectively bootstrapping and applying security patches and errata in one motion.

Notes

The bootstrap subcommand is generally only used once to prepare the system. The only other use case for the bootstrap command is when a new FreeBSD version is released and you want to start deploying containers on that version.

To update a release as patches are made available, see the *bastille update* command.

Downloaded artifacts are stored in the *bastille/cache/version* directory. “bootstrapped” releases are stored in *bastille/releases/version*.

To manually bootstrap a release (aka bring your own archive), place your archive in bastille/cache/name and extract to bastille/releases/name. Your mileage may vary; let me know what happens.

9.1.2 Templates

Bastille aims to integrate container automation into the platform while maintaining a simple, uncomplicated design. Templates are git repositories with automation definitions for packages, services, file overlays, etc.

To download one of these templates see the example below.

Example

```
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/nginx
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/mariadb-server
ishmael ~ # bastille bootstrap https://gitlab.com/bastillebsd-templates/python3
```

Tips

See the documentation on templates for more information on how they work and how you can create or customize your own. Templates are a powerful part of Bastille and facilitate full container automation.

Notes

If you don’t want to bother with git to use templates you can create them manually on the Bastille system and apply them.

Templates are stored in *bastille/templates/namespace/name*. If you’d like to create a new template on your local system, simply create a new namespace within the templates directory and then one for the template. This namespacing allows users and groups to have templates without conflicting template names.

Once you’ve created the directory structure you can begin filling it with template hooks. Once you have a minimum number of hooks (at least one) you can begin applying your template.

9.2 cmd

To execute commands within the container you can use *bastille cmd*.

```
ishmael ~ # bastille cmd folsom ps -auxw
[folsom]:
USER    PID %CPU %MEM   VSZ  RSS TT  STAT STARTED     TIME COMMAND
root 71464  0.0  0.0 14536 2000  -  IsJ   4:52PM 0:00.00 /usr/sbin/syslogd -ss
```

(continues on next page)

(continued from previous page)

```
root 77447  0.0  0.0 16632 2140  -  SsJ   4:52PM 0:00.00 /usr/sbin/cron -J 60 -s
root 80591  0.0  0.0 18784 2340  1  R+J   4:53PM 0:00.00 ps -auxw
```

9.3 clone

To clone a container and make a duplicate use the *bastille clone* sub-command..

```
ishmael ~ # bastille clone azkaban rikers ip
[azkaban]:
```

Syntax requires a name for the new container and an IP address assignment.

```
Usage: bastille clone [TARGET] [NEW_NAME] [IPADRESS].
```

9.4 console

This sub-command launches a login shell into the container. Default is password-less root login.

```
ishmael ~ # bastille console folsom
[folsom]:
root@folsom:~ #
```

At this point you are logged in to the container and have full shell access. The system is yours to use and/or abuse as you like. Any changes made inside the container are limited to the container.

9.5 convert

To convert a thin container to a thick container use *bastille convert*.

```
ishmael ~ # bastille convert azkaban
[azkaban]:
```

Syntax requires only the target container to convert.

```
Usage: bastille convert TARGET
```

9.6 cp

This command allows efficiently copying files from host to container(s).

```
ishmael ~ # bastille cp ALL /tmp/resolv.conf-cf etc/resolv.conf
[bastion]:

[unbound0]:

[unbound1]:
```

(continues on next page)

(continued from previous page)

```

[squid]:

[nginx]:

[folsom]:
```

Unless you see errors reported in the output the *cp* was successful.

9.7 create

Bastille create uses any available bootstrapped release to create a lightweight container system. To create a container simply provide a name, bootstrapped release and a private (rfc1918) IP address.

- name
- release
- ip
- interface (optional)

```
ishmael ~ # bastille create folsom 11.3-RELEASE 10.17.89.10 [interface]

RELEASE: 11.3-RELEASE.
NAME: folsom.
IP: 10.17.89.10.
```

This command will create a 11.3-RELEASE container assigning the 10.17.89.10 ip address to the new system.

I recommend using private (rfc1918) ip address ranges for your container. These ranges include:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Bastille does its best to validate the submitted ip is valid. This has not been thouroughly tested–I generally use the 10/8 range.

9.8 destroy

Containers can be destroyed and thrown away just as easily as they were created. Note: containers must be stopped before destroyed.

```
ishmael ~ # bastille stop folsom
[folsom]:
folsom: removed
```

```
ishmael ~ # bastille destroy folsom
Deleting Container: folsom.
Note: containers console logs not destroyed.
/usr/local/bastille/logs/folsom_console.log
```

9.9 edit

To edit container configuration use *bastille edit*.

```
ishmael ~ # bastille edit azkaban [filename]
```

Syntax requires a target an optional filename. By default the file edited will be *jail.conf*. Other common filenames are *fstab* or *rctl.conf*.

```
Usage: bastille edit TARGET
```

9.10 export

Exporting a container creates an archive or image that can be sent to a different machine to be imported later. These exported archives can be used as container backups.

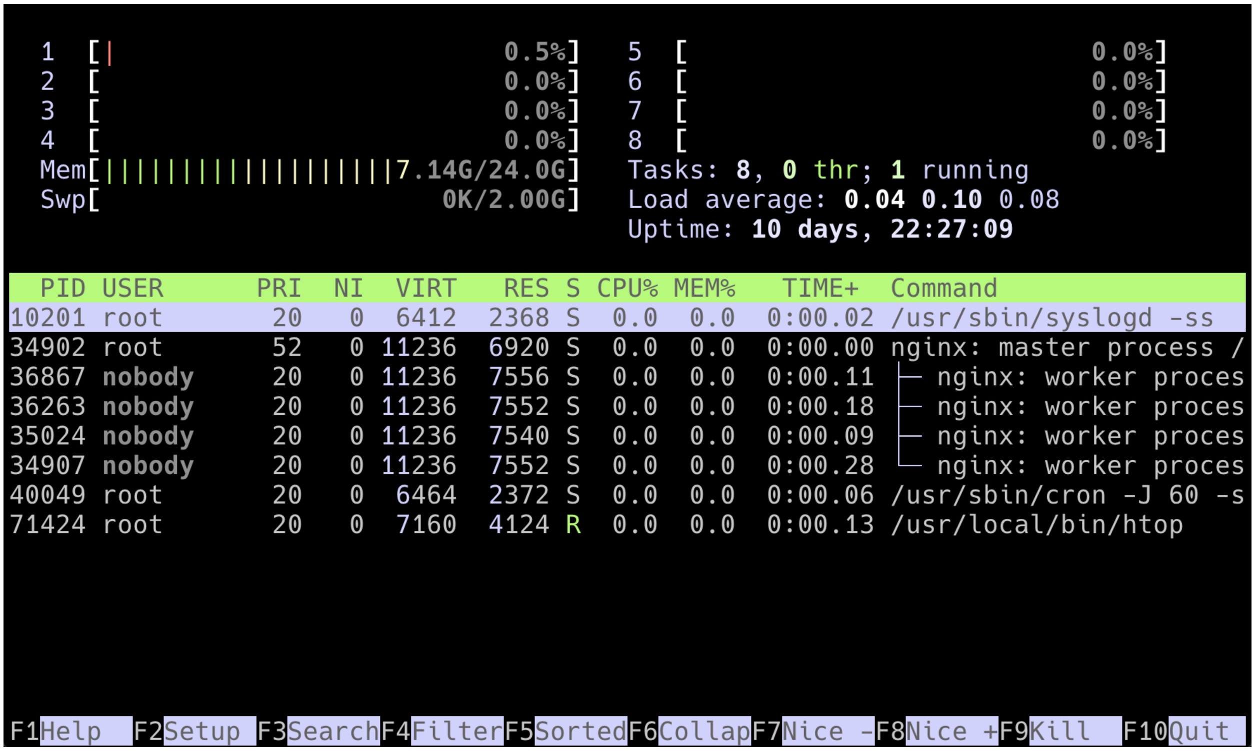
```
ishmael ~ # bastille export azkaban
```

The export sub-command supports both UFS and ZFS storage. ZFS based containers will use ZFS snapshots. UFS based containers will use *txz* archives.

```
Usage: bastille export TARGET
```

9.11 htop

This one runs *htop* inside the container. note: won't work if you don't have htop installed in the container.



```
  1  [|                          0.5%]   5  [                           0.0%]
  2  [                           0.0%]   6  [                           0.0%]
  3  [                           0.0%]   7  [                           0.0%]
  4  [                           0.0%]   8  [                           0.0%]
  Mem[||||||||||||||||||7.14G/24.0G]   Tasks: 8, 0 thr; 1 running
  Swp[                    0K/2.00G]   Load average: 0.04 0.10 0.08
                                       Uptime: 10 days, 22:27:09

  PID USER      PRI  NI  VIRT   RES S CPU% MEM%   TIME+  Command
10201 root       20   0  6412  2368 S  0.0  0.0  0:00.02 /usr/sbin/syslogd -ss
34902 root       52   0 11236  6920 S  0.0  0.0  0:00.00 nginx: master process /
36867 nobody     20   0 11236  7556 S  0.0  0.0  0:00.11 ├─ nginx: worker proces
36263 nobody     20   0 11236  7552 S  0.0  0.0  0:00.18 ├─ nginx: worker proces
35024 nobody     20   0 11236  7540 S  0.0  0.0  0:00.09 ├─ nginx: worker proces
34907 nobody     20   0 11236  7552 S  0.0  0.0  0:00.28 └─ nginx: worker proces
40049 root       20   0  6464  2372 S  0.0  0.0  0:00.06 /usr/sbin/cron -J 60 -s
71424 root       20   0  7160  4124 R  0.0  0.0  0:00.13 /usr/local/bin/htop





F1Help  F2Setup F3SearchF4FilterF5SortedF6CollapF7Nice -F8Nice +F9Kill  F10Quit
```

9.12 import

Import a container backup image or archive.

```
ishmael ~ # bastille import /path/to/archive.file
```

The import sub-command supports both UFS and ZFS storage. ZFS based containers will use ZFS snapshots. UFS based containers will use *txz* archives.

```
Usage: bastille import file [option]
```

9.13 mount

To mount storage within the container use *bastille mount*.

```
ishmael ~ # bastille mount azkaban /storage/foo /media/foo nullfs ro 0 0
[azkaban]:
```

Syntax follows standard */etc/fstab* format:

```
Usage: bastille mount TARGET host_path container_path [filesystem_type options dump␣
↪pass_number]
```

9.14 pkg

To manage binary packages within the container use *bastille pkg*.

```
ishmael ~ # bastille pkg folsom 'install vim-console git-lite zsh'
[folsom]:
The package management tool is not yet installed on your system.
Do you want to fetch and install it now? [y/N]: y
Bootstrapping pkg from pkg+http://pkg.FreeBSD.org/FreeBSD:10:amd64/quarterly, please␣
↪wait...
Verifying signature with trusted certificate pkg.freebsd.org.2013102301... done
[folsom] Installing pkg-1.10.5_5...
[folsom] Extracting pkg-1.10.5_5: 100%
Updating FreeBSD repository catalogue...
pkg: Repository FreeBSD load error: access repo file(/var/db/pkg/repo-FreeBSD.sqlite)␣
↪failed: No such file or directory
[folsom] Fetching meta.txz: 100%    944 B   0.9kB/s    00:01
[folsom] Fetching packagesite.txz: 100%    6 MiB   3.4MB/s    00:02
Processing entries: 100%
FreeBSD repository update completed. 32550 packages processed.
All repositories are up to date.
Updating database digests format: 100%
The following 10 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
    vim-console: 8.1.0342
    git-lite: 2.19.1
    zsh: 5.6.2
    expat: 2.2.6_1
```

(continues on next page)

(continued from previous page)

```
    curl: 7.61.1
    libnghttp2: 1.33.0
    ca_root_nss: 3.40
    pcre: 8.42
    gettext-runtime: 0.19.8.1_1
    indexinfo: 0.3.1

Number of packages to be installed: 10

The process will require 77 MiB more space.
17 MiB to be downloaded.

Proceed with this action? [y/N]: y
[folsom] [1/10] Fetching vim-console-8.1.0342.txz: 100%    5 MiB   5.8MB/s    00:01
[folsom] [2/10] Fetching git-lite-2.19.1.txz: 100%    4 MiB   2.1MB/s    00:02
[folsom] [3/10] Fetching zsh-5.6.2.txz: 100%    4 MiB   4.4MB/s    00:01
[folsom] [4/10] Fetching expat-2.2.6_1.txz: 100%  109 KiB 111.8kB/s    00:01
[folsom] [5/10] Fetching curl-7.61.1.txz: 100%    1 MiB   1.2MB/s    00:01
[folsom] [6/10] Fetching libnghttp2-1.33.0.txz: 100%  107 KiB 109.8kB/s    00:01
[folsom] [7/10] Fetching ca_root_nss-3.40.txz: 100%  287 KiB 294.3kB/s    00:01
[folsom] [8/10] Fetching pcre-8.42.txz: 100%    1 MiB   1.2MB/s    00:01
[folsom] [9/10] Fetching gettext-runtime-0.19.8.1_1.txz: 100%  148 KiB 151.3kB/s   ␣
↪00:01
[folsom] [10/10] Fetching indexinfo-0.3.1.txz: 100%    6 KiB   5.7kB/s    00:01
Checking integrity... done (0 conflicting)
[folsom] [1/10] Installing libnghttp2-1.33.0...
[folsom] [1/10] Extracting libnghttp2-1.33.0: 100%
[folsom] [2/10] Installing ca_root_nss-3.40...
[folsom] [2/10] Extracting ca_root_nss-3.40: 100%
[folsom] [3/10] Installing indexinfo-0.3.1...
[folsom] [3/10] Extracting indexinfo-0.3.1: 100%
[folsom] [4/10] Installing expat-2.2.6_1...
[folsom] [4/10] Extracting expat-2.2.6_1: 100%
[folsom] [5/10] Installing curl-7.61.1...
[folsom] [5/10] Extracting curl-7.61.1: 100%
[folsom] [6/10] Installing pcre-8.42...
[folsom] [6/10] Extracting pcre-8.42: 100%
[folsom] [7/10] Installing gettext-runtime-0.19.8.1_1...
[folsom] [7/10] Extracting gettext-runtime-0.19.8.1_1: 100%
[folsom] [8/10] Installing vim-console-8.1.0342...
[folsom] [8/10] Extracting vim-console-8.1.0342: 100%
[folsom] [9/10] Installing git-lite-2.19.1...
===> Creating groups.
Creating group 'git_daemon' with gid '964'.
===> Creating users
Creating user 'git_daemon' with uid '964'.
[folsom] [9/10] Extracting git-lite-2.19.1: 100%
[folsom] [10/10] Installing zsh-5.6.2...
[folsom] [10/10] Extracting zsh-5.6.2: 100%
```

The PKG sub-command can, of course, do more than just *install*. The expectation is that you can fully leverage the pkg manager. This means, *install*, *update*, *upgrade*, *audit*, *clean*, *autoremove*, etc., etc.

```
ishmael ~ # bastille pkg ALL upgrade
[bastion]:
Updating pkg.bastillebsd.org repository catalogue...
```

(continues on next page)

(continued from previous page)

```
[bastion] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[bastion] Fetching packagesite.txz: 100%  118 KiB 121.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (1 candidates): 100%
Processing candidates (1 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[unbound0]:
Updating pkg.bastillebsd.org repository catalogue...
[unbound0] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[unbound0] Fetching packagesite.txz: 100%  118 KiB 121.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[unbound1]:
Updating pkg.bastillebsd.org repository catalogue...
[unbound1] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[unbound1] Fetching packagesite.txz: 100%  118 KiB 121.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[squid]:
Updating pkg.bastillebsd.org repository catalogue...
[squid] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[squid] Fetching packagesite.txz: 100%  118 KiB 121.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (0 candidates): 100%
Processing candidates (0 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

[nginx]:
Updating pkg.bastillebsd.org repository catalogue...
[nginx] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[nginx] Fetching packagesite.txz: 100%  118 KiB 121.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 493 packages processed.
All repositories are up to date.
Checking for upgrades (1 candidates): 100%
Processing candidates (1 candidates): 100%
The following 1 package(s) will be affected (of 0 checked):
```

(continues on next page)

(continued from previous page)

```
Installed packages to be UPGRADED:
    nginx-lite: 1.14.0_14,2 -> 1.14.1,2

Number of packages to be upgraded: 1

315 KiB to be downloaded.

Proceed with this action? [y/N]: y
[nginx] [1/1] Fetching nginx-lite-1.14.1,2.txz: 100%  315 KiB 322.8kB/s    00:01
Checking integrity... done (0 conflicting)
[nginx] [1/1] Upgrading nginx-lite from 1.14.0_14,2 to 1.14.1,2...
===> Creating groups.
Using existing group 'www'.
===> Creating users
Using existing user 'www'.
[nginx] [1/1] Extracting nginx-lite-1.14.1,2: 100%
You may need to manually remove /usr/local/etc/nginx/nginx.conf if it is no longer␣
↪needed.
```

9.15 rdr

bastille rdr allows you to configure dynamic rdr rules for your containers without modifying pf.conf (assuming you are using the *bastille0* interface for a private network and have enabled *rdr-anchor 'rdr/*'* in /etc/pf.conf as described in the Networking section).

Note: you need to be careful if host services are configured to run on all interfaces as this will include the jail interface - you should sepcify the interface they run on in rc.conf (or other config files)

```
# bastille rdr --help
Usage: bastille rdr TARGET [clear] | [list] | [tcp <host_port> <jail_port>] | [udp
↪<host_port> <jail_port>]
# bastille rdr dev1 tcp 2001 22
# bastille rdr dev1 list
rdr on em0 inet proto tcp from any to any port = 2001 -> 10.17.89.1 port 22
# bastille rdr dev1 udp 2053 53
# bastille rdr dev1 list
rdr on em0 inet proto tcp from any to any port = 2001 -> 10.17.89.1 port 22
rdr on em0 inet proto udp from any to any port = 2053 -> 10.17.89.1 port 53
# bastille rdr dev1 clear
nat cleared
```

9.16 rename

Rename a container.

```
ishmael ~ # bastille rename azkaban arkham
```

```
Usage: bastille rename TARGET new_name
```

9.17 restart

To restart a container you can use the *bastille restart* command.

```
ishmael ~ # bastille restart folsom
[folsom]:
folsom: removed

[folsom]:
folsom: created
```

9.18 service

The *service* sub-command allows for managing services within containers. This allows you to start, stop, restart, and otherwise interact with services running inside the containers.

```
ishmael ~ # bastille service web01 'nginx start'
ishmael ~ # bastille service db01 'mysql-server restart'
ishmael ~ # bastille service proxy 'nginx configtest'
ishmael ~ # bastille service proxy 'nginx enable'
ishmael ~ # bastille service proxy 'nginx disable'
ishmael ~ # bastille service proxy 'nginx delete'
```

9.19 start

To start a container you can use the *bastille start* command.

```
ishmael ~ # bastille start folsom
[folsom]:
folsom: created
```

9.20 stop

To stop a container you can use the *bastille stop* command.

```
ishmael ~ # bastille stop folsom
[folsom]:
folsom: removed
```

9.21 sysrc

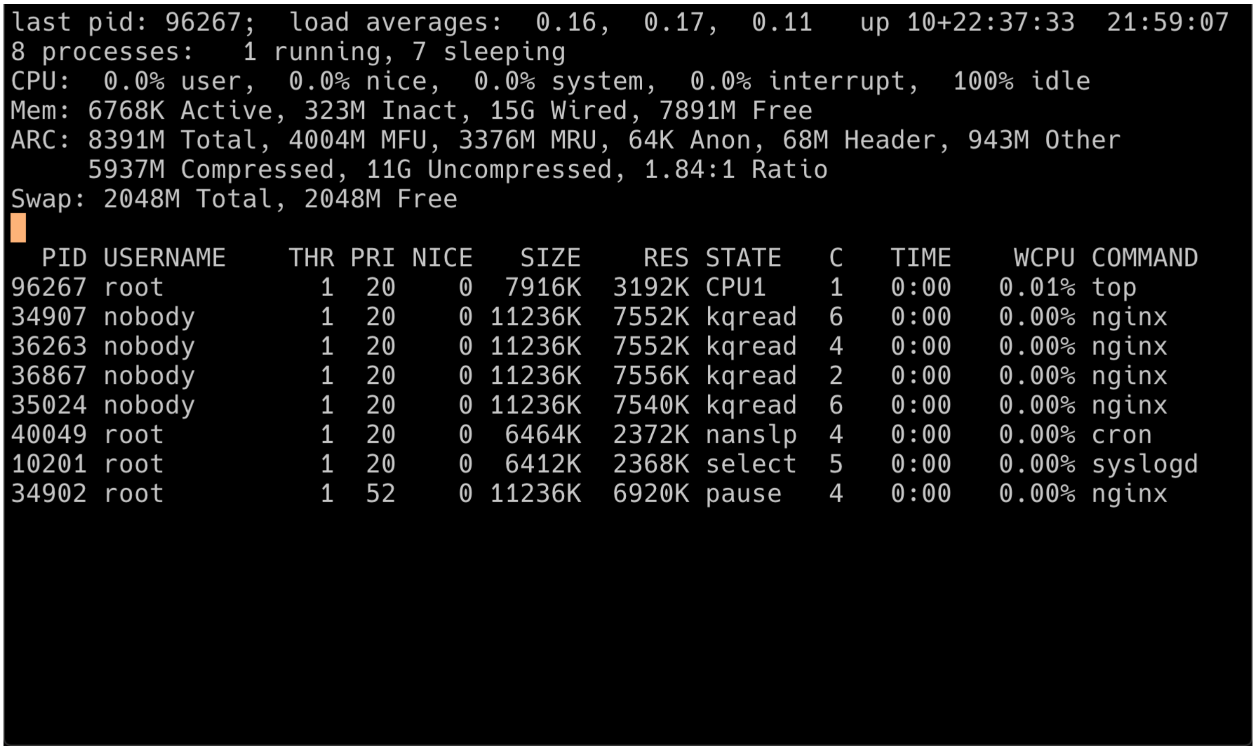
The *sysrc* sub-command allows for safely editing system configuration files. In container terms, this allows us to toggle on/off services and options at startup.

```
ishmael ~ # bastille sysrc nginx nginx_enable="YES"
[nginx]:
nginx_enable: NO -> YES
```

See *man sysrc(8)* for more info.

9.22 top

This one runs *top* in that container.



```
last pid: 96267;  load averages:  0.16,  0.17,  0.11   up 10+22:37:33  21:59:07
8 processes:   1 running, 7 sleeping
CPU:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
Mem: 6768K Active, 323M Inact, 15G Wired, 7891M Free
ARC: 8391M Total, 4004M MFU, 3376M MRU, 64K Anon, 68M Header, 943M Other
     5937M Compressed, 11G Uncompressed, 1.84:1 Ratio
Swap: 2048M Total, 2048M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
96267 root          1  20    0  7916K  3192K CPU1    1   0:00   0.01% top
34907 nobody        1  20    0 11236K  7552K kqread  6   0:00   0.00% nginx
36263 nobody        1  20    0 11236K  7552K kqread  4   0:00   0.00% nginx
36867 nobody        1  20    0 11236K  7556K kqread  2   0:00   0.00% nginx
35024 nobody        1  20    0 11236K  7540K kqread  6   0:00   0.00% nginx
40049 root          1  20    0  6464K  2372K nanslp  4   0:00   0.00% cron
10201 root          1  20    0  6412K  2368K select  5   0:00   0.00% syslogd
34902 root          1  52    0 11236K  6920K pause   4   0:00   0.00% nginx
```

9.23 umount

To unmount storage from a container use *bastille umount*.

```
ishmael ~ # bastille umount azkaban /media/foo
[azkaban]:
```

Syntax requires only the container path to unmount:

```
Usage: bastille umount TARGET container_path
```

9.24 update

The *update* command targets a release instead of a container. Because every container is based on a release, when the release is updated all the containers are automatically updated as well.

If no updates are available, a message will be shown:

```
ishmael ~ # bastille update 11.2-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 11.2-RELEASE from update4.freebsd.org... done.
Fetching metadata index... done.
Inspecting system... done.
Preparing to download files... done.

No updates needed to update system to 11.2-RELEASE-p4.
No updates are available to install.
```

The older the release, however, the more updates will be available:

```
ishmael ~ # bastille update 10.4-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 10.4-RELEASE from update1.freebsd.org... done.
Fetching metadata index... done.
Fetching 2 metadata patches.. done.
Applying metadata patches... done.
Fetching 2 metadata files... done.
Inspecting system... done.
Preparing to download files... done.

The following files will be added as part of updating to 10.4-RELEASE-p13:
...[snip]...
```

To be safe, you may want to restart any containers that have been updated live.

9.25 upgrade

This command lets you upgrade a release to a new release. Depending on the workflow this can be similar to a *bootstrap*.

```
ishmael ~ # bastille upgrade 12.0-RELEASE 12.1-RELEASE
```

9.26 verify

This command scans a bootstrapped release and validates that everything looks in order. This is not a 100% comprehensive check, but it compares the release against a “known good” index.

If you see errors or issues here, consider deleting and re-bootstrapping the release.

```
ishmael ~ # bastille verify 11.2-RELEASE
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 11.2-RELEASE from update1.freebsd.org... done.
Fetching metadata index... done.
Fetching 1 metadata patches. done.
Applying metadata patches... done.
Fetching 1 metadata files... done.
Inspecting system... done.
```

CHAPTER 10

Template

Looking for ready made CI/CD validated [Bastille Templates](https://gitlab.com/BastilleBSD-Templates)?

Bastille supports a templating system allowing you to apply files, pkgs and execute commands inside the containers automatically.

Currently supported template hooks are: *LIMITS*, *INCLUDE*, *PRE*, *FSTAB*, *PKG*, *OVERLAY*, *SYSRC*, *SERVICE*, *CMD*.

Templates are created in *${bastille_prefix}/templates* and can leverage any of the template hooks.

10.1 Bastille 0.7.x

Bastille 0.7.x introduces a template syntax that is more flexible and allows any-order scripting. Previous versions had a hard template execution order and instructions were spread across multiple files. The new syntax is done in a *Bastillefile* and the template hook (see below) files are replaced with template hook commands.

10.2 Template Automation Hooks

HOOK	format	example
LIMITS	resource value	memoryuse 1G
INCLUDE	template path/URL	http?://TEMPLATE_URL or project/path
PRE	/bin/sh command	mkdir -p /usr/local/my_app/html
FSTAB	fstab syntax	/host/path container/path nullfs ro 0 0
PKG	port/pkg name(s)	vim-console zsh git-lite tree htop
OVERLAY	path(s)	etc root usr (one per line)
SYSRC	sysrc command(s)	nginx_enable=YES
SERVICE	service command	'nginx start' OR 'postfix reload'
CMD	/bin/sh command	/usr/bin/chsh -s /usr/local/bin/zsh

Note: SYSRC requires that NO quotes be used or that quotes (“) be escaped ie; (″)

Place these uppercase template hook commands into a *Bastillefile* in any order and automate container setup as needed.

In addition to supporting template hooks, Bastille supports overlaying files into the container. This is done by placing the files in their full path, using the template directory as “/”.

An example here may help. Think of *bastille/templates/username/template*, our example template, as the root of our filesystem overlay. If you create an *etc/hosts* or *etc/resolv.conf* *inside* the template directory, these can be overlayed into your container.

Note: due to the way FreeBSD segregates user-space, the majority of your overlayed template files will be in *usr/local*. The few general exceptions are the *etc/hosts*, *etc/resolv.conf*, and *etc/rc.conf.local*.

After populating *usr/local* with custom config files that your container will use, be sure to include *usr* in the template OVERLAY definition. eg;

```
echo "usr" > /usr/local/bastille/templates/username/template/OVERLAY
```

The above example “usr” will include anything under “usr” inside the template. You do not need to list individual files. Just include the top-level directory name. List these top-level directories one per line.

10.3 Applying Templates

Containers must be running to apply templates.

Bastille includes a *template* command. This command requires a target and a template name. As covered in the previous section, template names correspond to directory names in the *bastille/templates* directory.

```
ishmael ~ # bastille template ALL username/template
[proxy01]:
Copying files...
Copy complete.
Installing packages.
pkg already bootstrapped at /usr/local/sbin/pkg
vulnxml file up-to-date
0 problem(s) in the installed packages found.
Updating bastillebsd.org repository catalogue...
[cdn] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[cdn] Fetching packagesite.txz: 100%  121 KiB 124.3kB/s    00:01
Processing entries: 100%
bastillebsd.org repository update completed. 499 packages processed.
All repositories are up to date.
Checking integrity... done (0 conflicting)
The most recent version of packages are already installed
Updating services.
cron_flags: -J 60 -> -J 60
sendmail_enable: NONE -> NONE
syslogd_flags: -ss -> -ss
Executing final command(s).
chsh: user information updated
Template Complete.

[web01]:
Copying files...
Copy complete.
Installing packages.
```

(continues on next page)

(continued from previous page)

```
pkg already bootstrapped at /usr/local/sbin/pkg
vulnxml file up-to-date
0 problem(s) in the installed packages found.
Updating pkg.bastillebsd.org repository catalogue...
[poudriere] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[poudriere] Fetching packagesite.txz: 100%  121 KiB 124.3kB/s    00:01
Processing entries: 100%
pkg.bastillebsd.org repository update completed. 499 packages processed.
Updating bastillebsd.org repository catalogue...
[poudriere] Fetching meta.txz: 100%    560 B   0.6kB/s    00:01
[poudriere] Fetching packagesite.txz: 100%  121 KiB 124.3kB/s    00:01
Processing entries: 100%
bastillebsd.org repository update completed. 499 packages processed.
All repositories are up to date.
Checking integrity... done (0 conflicting)
The most recent version of packages are already installed
Updating services.
cron_flags: -J 60 -> -J 60
sendmail_enable: NONE -> NONE
syslogd_flags: -ss -> -ss
Executing final command(s).
chsh: user information updated
Template Complete.
```

Note: FreeBSD introduced container technology twenty years ago, long before the industry standardized on the term “container”. Internally, FreeBSD refers to these containers as “jails”.

CHAPTER 11

jail.conf

In this section we'll look at the default config for a new container. The defaults are sane for most applications, but if you want to tweak the settings here they are.

A *jail.conf* template is used each time a new container is created. This template looks like this:

```
{name} {
  devfs_ruleset = 4;
  enforce_statfs = 2;
  exec.clean;
  exec.consolelog = /var/log/bastille/{name}_console.log;
  exec.start = '/bin/sh /etc/rc';
  exec.stop = '/bin/sh /etc/rc.shutdown';
  host.hostname = {name};
  interface = {interface};
  mount.devfs;
  mount.fstab = /usr/local/bastille/jails/{name}/fstab;
  path = /usr/local/bastille/jails/{name}/root;
  securelevel = 2;

  ip4.addr = x.x.x.x;
  ip6 = disable;
}
```

11.1 devfs_ruleset

```
devfs_ruleset
  The number of the devfs ruleset that is enforced for mounting
  devfs in this jail.  A value of zero (default) means no ruleset
  is enforced.  Descendant jails inherit the parent jail's devfs
  ruleset enforcement.  Mounting devfs inside a jail is possible
  only if the allow.mount and allow.mount.devfs permissions are
  effective and enforce_statfs is set to a value lower than 2.
```

(continues on next page)

(continued from previous page)

```
  Devfs rules and rulesets cannot be viewed or modified from inside
  a jail.

  NOTE: It is important that only appropriate device nodes in devfs
  be exposed to a jail; access to disk devices in the jail may
  permit processes in the jail to bypass the jail sandboxing by
  modifying files outside of the jail.  See devfs(8) for
  information on how to use devfs rules to limit access to entries
  in the per-jail devfs.  A simple devfs ruleset for jails is
  available as ruleset #4 in /etc/defaults/devfs.rules.
```

11.2 enforce_statfs

```
enforce_statfs
  This determines what information processes in a jail are able to
  get about mount points.  It affects the behaviour of the
  following syscalls: statfs(2), fstatfs(2), getfsstat(2), and
  fhstatfs(2) (as well as similar compatibility syscalls).  When
  set to 0, all mount points are available without any
  restrictions.  When set to 1, only mount points below the jail's
  chroot directory are visible.  In addition to that, the path to
  the jail's chroot directory is removed from the front of their
  pathnames.  When set to 2 (default), above syscalls can operate
  only on a mount-point where the jail's chroot directory is
  located.
```

11.3 exec.clean

```
exec.clean
  Run commands in a clean environment.  The environment is
  discarded except for HOME, SHELL, TERM and USER.  HOME and SHELL
  are set to the target login's default values.  USER is set to the
  target login.  TERM is imported from the current environment.
  The environment variables from the login class capability
  database for the target login are also set.
```

11.4 exec.consolelog

```
exec.consolelog
  A file to direct command output (stdout and stderr) to.
```

11.5 exec.start

```
exec.start
  Command(s) to run in the jail environment when a jail is created.
  A typical command to run is "sh /etc/rc".
```

11.6 exec.stop

```
exec.stop
  Command(s) to run in the jail environment before a jail is
  removed, and after any exec.prestop commands have completed.  A
  typical command to run is "sh /etc/rc.shutdown".
```

11.7 host.hostname

```
host.hostname
  The hostname of the jail.  Other similar parameters are
  host.domainname, host.hostuuid and host.hostid.
```

11.8 interface

```
interface
  A network interface to add the jail's IP addresses (ip4.addr and
  ip6.addr) to.  An alias for each address will be added to the
  interface before the jail is created, and will be removed from
  the interface after the jail is removed.
```

11.9 mount.devfs

```
mount.devfs
  Mount a devfs(5) filesystem on the chrooted /dev directory, and
  apply the ruleset in the devfs_ruleset parameter (or a default of
  ruleset 4: devfsrules_jail) to restrict the devices visible
  inside the jail.
```

11.10 mount.fstab

```
mount.fstab
  An fstab(5) format file containing filesystems to mount before
  creating a jail.
```

11.11 path

```
path
  The directory which is to be the root of the jail.  Any commands
  run inside the jail, either by jail or from jexec(8), are run
  from this directory.
```

11.12 securelevel

By default, Bastille containers run at *securelevel = 2;*. See below for the implications of kernel security levels and when they might be altered.

Note: Bastille does not currently have any mechanism to automagically change securelevel settings. My recommendation is this only be altered manually on a case-by-case basis and that “Highly secure mode” is a sane default for most use cases.

```
The kernel runs with five different security levels.  Any super-user
process can raise the level, but no process can lower it.  The security
levels are:

-1    Permanently insecure mode - always run the system in insecure mode.
      This is the default initial value.

0     Insecure mode - immutable and append-only flags may be turned off.
      All devices may be read or written subject to their permissions.

1     Secure mode - the system immutable and system append-only flags may
      not be turned off; disks for mounted file systems, /dev/mem and
      /dev/kmem may not be opened for writing; /dev/io (if your platform
      has it) may not be opened at all; kernel modules (see kld(4)) may
      not be loaded or unloaded.  The kernel debugger may not be entered
      using the debug.kdb.enter sysctl.  A panic or trap cannot be forced
      using the debug.kdb.panic and other sysctl's.

2     Highly secure mode - same as secure mode, plus disks may not be
      opened for writing (except by mount(2)) whether mounted or not.
      This level precludes tampering with file systems by unmounting
      them, but also inhibits running newfs(8) while the system is multi-
      user.

      In addition, kernel time changes are restricted to less than or
      equal to one second.  Attempts to change the time by more than this
      will log the message "Time adjustment clamped to +1 second".

3     Network secure mode - same as highly secure mode, plus IP packet
      filter rules (see ipfw(8), ipfirewall(4) and pfctl(8)) cannot be
      changed and dummynet(4) or pf(4) configuration cannot be adjusted.
```

CHAPTER 12

Copyright

Note: this documentation is included with the source code in *docs*.