MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

BHYVEARM64: VIRTUALIZATION ON ARMV8-A

ILLUMOS CONTAINERS USING OMNIOSCE

ISCSI ON FREEBSD

HTTP/2 AND PHP WITH APACHE ON FREEBSD
NOT AS SIMPLE AS IT SEEMED

REDUNDANT FIREWALLS WITH OPENBSD, CARP AND PFSYNC
BY DANIELE MAZZOCCHIO

JUST TAKES 5 SECONDS TO GROW YOUR TEAM CULTURE
BY E.G.NADHAN

INTERVIEW WITH JOEL KNIGHT

VOL 12 NO 08
ISSUE 08/2018 (108)
ISSN 1898-9144

IS AFFORDABLE FLASH STORAGE OUT OF REACH?

NOT ANYMORE!

IXSYSTEMS DELIVERS A FLASH ARRAY FOR UNDER $10,000.

Introducing FreeNAS® Certified Flash: A high performance all-flash array at the cost of spinning disk.

- Unifies NAS, SAN, and object storage to support multiple workloads
- Runs FreeNAS, the world's #1 software-defined storage solution
- Performance-oriented design provides maximum throughput/IOPs and lowest latency
- OpenZFS ensures data integrity
- Perfectly suited for Virtualization, Databases, Analytics, HPC, and M&E
- 10TB of all-flash storage for less than $10,000
- Maximizes ROI via high-density SSD technology and inline data reduction
- Scales to 100TB in a 2U form factor

The all-flash datacenter is now within reach. Deploy a FreeNAS Certified Flash array today from iXsystems and take advantage of all the benefits flash delivers.

Call or click today! **1-855-GREP-4-IX** (US) **|** **1-408-943-4100** (Non-US) **|** **www.iXsystems.com/FreeNAS-certified-servers**

DON'T DEPEND ON CONSUMER-GRADE STORAGE.

intel inside™

KEEP YOUR DATA SAFE!

Buy from amazon.com

USE AN ENTERPRISE-GRADE STORAGE SYSTEM FROM IXSYSTEMS INSTEAD.

The FreeNAS Mini: Plug it in and boot it up — it just works.

- Runs FreeNAS, the world's #1 software-defined storage solution
- Unifies NAS, SAN, and object storage to support multiple workloads
- Encrypt data at rest or in flight using an 8-Core 2.4GHz Intel® Atom® processor
- OpenZFS ensures data integrity
- A 4-bay or 8-bay desktop storage array that scales to 48TB and packs a wallop
- Backed by a 1 year parts and labor warranty, and supported by the Silicon Valley team that designed and built it
- Perfectly suited for SoHo/SMB workloads like backups, replication, and file sharing
- Lowers storage TCO through its use of enterprise-class hardware, ECC RAM, optional flash, white-glove support, and enterprise hard drives

And really — why would you trust storage from anyone else?

iXsystems™

Call or click today! **1-855-GREP-4-IX** (US) **|** **1-408-943-4100** (Non-US) **|** **www.iXsystems.com/Freenas-Mini** or purchase on Amazon.

Editor's Word

Dear Readers,

Summer is here! A season which appeals to affection and sentiments, and of course, vacation time isn't complete without a touch of laziness. As Ella sang: '*Summertime, and the livin' is easy',* it's indeed a good time. I hope you will work on unfinished projects, explore the latest technologies, and encounter numerous exciting tasks. Here, we concertedly continue to publish great lectures for you and take pride in them as well. Therefore, as you'll be enjoying a sunset view on your porch, have a quick read at our BSD Magazine issue to crown the special day.

Now, let's have a glimpse of what our experts prepared for you. Every month, Rob Somerville and, E.G. Nathan share interesting ideas and thoughts regarding the world of technology. In fact, we are privileged to be the platform where some of these great authors choose to share their experiences with you. So, don't miss out on their great insights through their columns and articles for this month. For the FreeBSD enthusiasts, Abdorrahman Homaei has given additional tips from his galore of new tricks in his article. It's due to his consistency that we have published over 15 articles so that you can learn more about FreeBSD.

In this issue, you will also find more articles written by Bob Cromwell. This issue also includes an article by Carlos Neira, who is a household name for this Magazine. He doesn't only produce excellent technical articles for the BSD Magazine but also acts as the instructor for our courses.

I am also delighted to introduce Alexandru Elisei as a man of vast knowledge in his field. This BSD Magazine issue will feature his first submission, and I am confident that you will love it. Additionally, this issue offers you a chance to know more about our experts on a personal and professional level. So make sure to read the interview with Joel Knight. Last, but not the least, meet a great expert and blogger, Daniele Mazzocchio. He is a great teacher with impressive knowledge, and you can learn more on his blog: www.kernel-panic.it.

I would also like to mention that I issued the course ebook: *Improve Your PostgreSQL Skills* by Luca Ferrari and next week I plan to publish the course ebook: *Device Driver Development for BSD* by Rafael Santiago. Both eBooks will help you expand your knowledge and learn more about the said subjects.

So let's read! If any questions arise in your mind during or after reading the articles, please feel free to contact me via email: *ewa@bsdmag.org*. We hope you enjoy reading this issue and develop new skills with our magazine!

Thank you,

Ewa & The BSD team

Table of Contents

Online shopping and electronic transactions are revolutionizing the way business is being carried out, both for individuals and corporate entities. Are we entering a golden age of choice, or should the Latin phrase Caveat Emptor be embedded on every "accept" button for Internet sales?

Rob Somerville

I've just been ripped off of £153.25 for a Samsung Galaxy J5 mobile phone or to be more accurate, Amazon has, along with approximately 1,000 other customers who have paid exorbitant amounts of money to a clearly fraudulent storefront that has exploited a subtle flaw in the E-commerce model that Amazon, eBay, and PayPal operate.

MAGAZINE BSD

Editor in Chief:

Ewa Dudzic
ewa@bsdmag.org
www.bsdmag.org

Contributing:

Sanel Zukan, Luca Ferrari, José B. Alós, Carlos Klop, Eduardo Lavaque, Jean-Baptiste Boric, Rafael Santiago, Andrey Ferriyan, Natalia Portillo, E.G Nadhan, Daniel Cialdella Converti, Vitaly Repin, Henrik Nyh, Renan Dias, Rob Somerville, Hubert Feyrer, Kalin Staykov, Manuel Daza, Abdorrahman Homaei, Amit Chugh, Mohamed Farag, Bob Cromwell, David Rodriguez, Carlos Antonio Neira Bustos, Antonio Francesco Gentile, Randy Remirez, Vishal Lambe, Mikhail Zakharov, Pedro Giffuni, David Carlier, Albert Hui, Marcus Shmitt, Aryeh Friedman

Top Betatesters & Proofreaders:

Daniel Cialdella Converti, Eric De La Cruz Lugo, Daniel LaFlamme, Steven Wierckx, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Katherine Dizon, Natalie Fahey, and Mark VonFange.

Special Thanks:

Denise Ebery
Katherine Dizon

Senior Consultant/Publisher: Paweł Marciniak

Publisher: Hakin9 Media SK,
02-676 Warsaw, Poland Postepu 17D, Poland
worldwide publishing
editors@bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org

DigitalOcean Introduces Kubernetes Product for Simple, Scalable Container Deployment and Orchestration

DigitalOcean, the cloud platform for developers and their teams, today announced its DigitalOcean Kubernetes product, the easiest way to run containerized applications in the cloud. Designed for developers and businesses who want a simple way to deploy and manage container workloads, DigitalOcean Kubernetes removes the headache involved in setting up, managing and securing Kubernetes clusters while incorporating DigitalOcean's trademark simplicity and ease of use.

"Over the last year, Kubernetes has emerged as the container orchestration platform of choice, and as one of the leading public clouds, investing in supporting our customers' adoption of containers was a natural evolution to our roadmap," said DigitalOcean VP of Product Shiven Ramji. "We've always been devoted to providing simple solutions for developers — starting with our cloud servers, Droplets. This product is no exception, allowing developers to focus on successfully shipping their applications while not being burdened by the complexity involved with creating and running a highly scalable and secure cluster across multiple apps."

The application container market is estimated to grow to $2.7B by 2020, according to 451 Research. Further, developers and those in DevOps are growing more committed to Kubernetes: in 2016, just under half said they were committed to the system but by 2017, 77 percent said the same, according to the Cloud Native Computing Foundation. Despite Kubernetes' growing popularity, on its own, it can be complex for developers to manage.

By offering Kubernetes integrated with DigitalOcean's core product suite — which includes Compute Servers, Block Storage, Object Storage, Firewalls, Load Balancers and more — businesses will have the freedom to run their existing workloads on DigitalOcean without special configuration. Key features and benefits of DigitalOcean Kubernetes include:

Dedicated Managed Kubernetes Cluster: Each customer receives their own cluster, which provides security and isolation for their containerized applications with access to the full Kubernetes API.

Integrated Storage Scalability: DigitalOcean products for block storage and object storage are built in, providing storage for any amount of data.

Included Security: Cloud Firewalls are included, making it easy to manage network traffic in and out of the Kubernetes cluster. Additionally, DigitalOcean will provide cluster security scanning capabilities to alert users of flaws and vulnerabilities.

Continuous Delivery: Simple integration with popular continuous integration services; developers can easily set up a full continuous delivery pipeline in two clicks, providing faster and more robust rollout of new application functionality.

Team Management: Kubernetes deployments can be a large team effort. DigitalOcean's "teams" feature allows development teams to manage access and permissions to the cluster easily.

Extended Insights: In typical Kubernetes environments, metrics, logs and events can be lost if nodes are spun down. To help developers learn from the performance of past environments, the DigitalOcean

Kubernetes product will store this information separate from the node indefinitely.
One-click Integrations: Similar to the existing one-click setups and integrations for Droplets, the product includes one-click integrations to deploy an entire application stack so developers can focus on solving their business problems and worry less about their Kubernetes cluster setup.
DigitalOcean Kubernetes will be available through an early access program starting in June with general availability planned for later this year. Sign up for early access at http://do.co/k8s.

Source: https://www.digitalocean.com/press/releases/digitalocean-introduces-kubernetes-product/

June 19 is FreeBSD Day!

June 19 has been declared FreeBSD Day. Join us in honoring The FreeBSD Project's pioneering legacy and continuing impact on technology.

What is FreeBSD?

FreeBSD is an open-source operating system developed out of the University of California at Berkley in 1993. Used by millions of people around the globe, FreeBSD is used to teach operating system concepts in universities. Companies also develop products on FreeBSD, and universities use it as a research platform.
In fact, there's a good chance you're already using at least some code derived from FreeBSD in your everyday life. For example, if you stream movies via Netflix, chat with friends on WhatsApp, or play the latest PlayStation 4 game sensation, you're already using FreeBSD.
As a pioneer in open-source technology, FreeBSD can be modified and redesigned to meet the needs of the user, free of charge within the guidelines of the license.

Why June 19th

June 19, 1993 was the day the official name for FreeBSD was agreed upon. See part of the email thread on the original post.
If you love FreeBSD, celebrate the 25th anniversary of your favorite open source operating system by doing the following:

- Introducing someone to FreeBSD or hosting an Installfest with your local meetup group Slides and materials for hosting a FreeBSD installfest.
- From TrueOS to FreeBSD on Virtual Box to Installing Ports, you can find a number of how-tos online.
- Check out the list of companies using and products based on FreeBSD.
- Helping to promote the day by printing and distributing the poster.
- Sending us stories of how your company uses FreeBSD to great success
  Telling us why you love FreeBSD using #FreeBSDDay on your Facebook, Twitter, and Instagram posts
- Consider donating to the Foundation to help us continue our support of the Project.

- Check out and share BSDNow's 6 hour retrospective of interviews from the FreeBSD community including an interview with Dr. Kirk McKusick from BSDCan 2018.

We look forward to commemorating the 25th anniversary of our favorite open-source operating system on June 19, and we hope you'll join us!

Source: https://www.freebsdfoundation.org/national-freebsd-day/

TrueOS to Focus on Core Operating System

The TrueOS Project has some big plans in the works, and we want to take a minute and share them with you. Many have come to know TrueOS as the "graphical FreeBSD" that makes things easy for newcomers to the BSDs. Today, we're announcing that TrueOS is shifting our focus a bit to become a cutting-edge operating system that keeps all of the stability that you know and love from ZFS (OpenZFS) and FreeBSD, and adds additional features to create a fresh, innovative operating system. Our goal is to create a core-centric operating system that is modular, functional, and perfect for do-it-yourselfers and advanced users alike.

TrueOS will become a downstream fork that will build on FreeBSD by integrating new software technologies like OpenRC and LibreSSL. Work has already begun which allows TrueOS to be used as a base platform for other projects, including JSON-based manifests, integrated Poudriere / pkg tools and much more. We're planning on a six-month release cycle to keep development moving and fresh, allowing us to bring you hot new features to ZFS, bhyve and related tools promptly. This makes TrueOS the perfect fit to serve as the basis for building other distributions.

Some of you are probably asking yourselves "But what if I want to have a graphical desktop?" Don't worry! We're making sure that everyone who knows and loves the legacy desktop version of TrueOS will be able to continue using a FreeBSD-based, graphical operating system in the future. For instance, if you want to add KDE, just use **sudo pkg install kde** and voila! You have your new shiny desktop. Easy right? This allows us to get back to our roots of being a desktop agnostic operating system. If you want to add a new desktop environment, you get to pick the one that best suits your use.

We know that some of you will still be looking for an out-of-the-box solution similar to legacy PC-BSD and TrueOS. We're happy to announce that Project Trident will take over graphical FreeBSD development going forward. Not much is going to change in that regard other than a new name! You'll still have Lumina Desktop as a lightweight and feature-rich desktop environment and tons of utilities from the legacy TrueOS toolchain like sysadm and AppCafe. There will be migration paths available for those that would like to move to other FreeBSD-based distributions like Project Trident or GhostBSD.

Source: https://www.trueos.org/blog/trueosdownstream/

Groundbreaking TrueNAS® M Series

intel XEON SILVER inside™

DISRUPTING STORAGE AT WARP SPEED

LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit **iXsystems.com/TrueNAS** or call **(855) GREP-4-iX** today!

HGST a Western Digital brand

iX systems™

Illumos Containers Using OmniOSce

Containers have been around almost two decades starting with FreeBSD jails implementation around 2000. Thereafter, Sun Microsystems took a step further and implemented Solaris Zones around 2004, which was based on FreeBSD's jails. Both containerization technologies allow you to partition your machine further and give you more mileage for your money as it is lighter than hardware virtualization. This means performance is better as applications run on bare metal. Enhanced security such that if a zone or jail is compromised, the attacker is confined to that virtual host. We will learn about Solaris Zones using an Illumos derivative called OmniOSce, and instructions that could be applied to other Illumos based distributions.

What you will learn:

- *What is a zone.*
- *Create and configure a zone.*
- *Define resource limits for a zone.*

What you should know:

- *Command line familiarity.*

What you will need:

- *OmniOSce bloody version or the latest LTS.*

What is a Zone?

A Solaris Container or Zone is the operating system-level virtualization.This abstraction makes processes in a Zone to believe that they are running on their kernel copy, but in reality, there is only one kernel doing all the work. For this abstraction to be useful resource, control facilities exist along with means for the zone to access hardware devices that are in the global zone. A global zone is one where all processes for all zones are available to peak into. Additionally,the global zone is where nonglobal zones are created, commonly referred to as GZ and NGZ.

Operations on NGZ like creation, deletion, configuration, booting, and halting are all done using two tools called zonecfg and zoneadm.

ZONECFG(1M)

This tool allows you to create or edit a zone definition. A zone is defined by the available resources it will have such as network interfaces, file systems, CPUs , devices, security constraints, brand, memory, and the resource control rules on the zone i.e., memory capping, maximum of processes allowed, etc.

```
neirac@krondor:~/zcage$ zonecfg help
Commands:

add <resource-type>
        (global scope)
add <property-name> <property-value>
        (resource scope)
        Add specified resource to configuration.

cancel
        Cancels resource/property specification.

clear <property-name>
        Clears property values.

commit
        Commits current configuration.  Configuration must be committed to
        be used by zoneadm.  Until the configuration is committed, changes
        can be removed with the revert command.  This operation is
        attempted automatically upon completion of a zonecfg session.

create [-F] [ -a <path> | -b | -t <template> ]
        Creates a configuration for the specified zone.  create should be
        used to begin configuring a new zone.  If overwriting an existing
        configuration, the -F flag can be used to force the action.  If
        -t template is given, creates a configuration identical to the
        specified template, except that the zone name is changed from
        template to zonename.  'create -a' creates a configuration from a
        detached zonepath.  'create -b' results in a blank configuration.
        'create' with no arguments applies the Sun default settings.

delete [-F]
        Deletes the specified zone.  The -F flag can be used to force the
        action.
```

ZONEADM (1M)

As the name implies, this tool allows you to administer a zone, so you could perform actions like start a zone, reboot it, stop it, or clone it. For more in-depth information, refer to the man page.

```
neirac@krondor:~/zcage$ zoneadm help
usage:  zoneadm help
        zoneadm [-z <zone>] [-u <uuid-match>] list
        zoneadm {-z <zone>|-u <uuid-match>} <subcommand>

Subcommands:

help
        Print usage message.

boot [-- boot_arguments]
        Activates (boots) specified zone.  See zoneadm(1m) for valid boot
        arguments.

halt
        Halts specified zone, bypassing shutdown scripts and removing runtime
        resources of the zone.

ready
        Prepares a zone for running applications but does not start any user
        processes in the zone.

shutdown [-r [-- boot_arguments]]
        Gracefully shutdown the zone or reboot if the '-r' option is specified.
        See zoneadm(1m) for valid boot arguments.

reboot [-- boot_arguments]
        Restarts the zone (equivalent to a halt / boot sequence).
        Fails if the zone is not active.  See zoneadm(1m) for valid boot
        arguments.
```

To list our available zones, we use zoneadm and the list operation.

```
neirac@krondor:~/zcage$ zoneadm list -icv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              ipkg     shared
   1 nodejs           running    /zones/nodejs                  sparse   excl
   - test04           configured /zones/test04                  sparse   excl
   - test05           installed  /zones/test05                  sparse   excl
```

Here is the explanation of the columns in this output:

- **ID -** A number representing the global zone ID, which is always 0.
- **STATUS -** A configured status reflects that the zonecfg has introduced the zone definition and the zone is ready to be installed. An installed status implies that it is ready to boot the zone and transition to the running state.
- **PATH -** The path where the zone data set resides.
- **BRAND -** A brand is a configured zone. It manages how a zone is set up during installation, and which application a system call table implements. In the LX brand, the Linux system call table allows running of unmodified Linux applications in a zone. The types of zones that are available in Omniosce are ipkg, lipkg, sparse, and lx. Let’s give a brief explanation of each type of brand.

- **ipkg -** The ipkg brand has a complete independent filesystem from the host system (global zone).
- **sparse -** The sparse only requires /etc to be generated at installation as it uses a read-only loopback of filesystems from the global zone. Thus, you could run many sparse zones in a modest machine since it consumes low disk space.
- **lipkg -** This is a Linked image zone. It links the packages in a zone to the global zone. If you update the global zone's packages, the linked-image zones get updated alongside it.

Creating and configuring a zone

Let's start by creating a zone definition. To do this, we will use zonecfg.

We will create a sparse branded zone with a filesystem stored in the /zones/test zfs dataset, and it will have its own IP address. We will use the virtual network interface called test0.

Type the following as root or with an account that has a Primary Administrator role.

First, we need to install the sparse brand in OmniOSce.

```
# pkg install brand/sparse
```

Next, we need to create the virtual network interface (*vnic*) that the zone will use.

```
# dladm create-vnic -l igb0 vnic0
```

Also, let's create a dataset on where our zones will live (replace *rpool* with your own pool).

```
# zfs create -o mountpoint=/zones
rpool/zones
```

We are now ready to create our zone.

```
# zonecfg -z test

test: No such zone configured
Use 'create' to begin configuring a new
zone.

zonecfg:test> create

zonecfg:test> set brand=sparse

zonecfg:test> set zonepath=/zones/test

zonecfg:test> set ip-type=exclusive

zonecfg:test> add net

zonecfg:test:net> set physical=vnic0

zonecfg:test:net> end

zonecfg:test> verify

zonecfg:test> commit

zonecfg:test> exit
```

If all went well, the zone would be in configured status (you can check the current state with a *zoneadm list -icv*)

Now, let's install it

```
# zoneadm -z test0 install
```

Next, please see Figure 1. After this, the zone is ready, and we can boot it and setup networking.

```
# zoneadm -z test boot

# zlogin test

# ipadm create-if vnic0

# ipadm create-addr -T static -a
local=IP_FOR_YOUR_ZONE/y vnic0/v4

# echo YOUR_GATEWAY_IP >
/etc/defaultrouter

# echo 'nameserver 8.8.8.8' >
/etc/resolv.conf

# cp /etc/nsswitch.{dns,conf}

# svcadm restart routing-setup

# ping google.cl

# google.cl is alive
```

```
A ZFS file system has been created for this zone.
Caching catalogs ... Done
       Image: Preparing at /zones/test02/root.
Sanity Check: Looking for 'entire' incorporation.
 Startup: Refreshing catalog 'extra.omnios' ... Done
 Startup: Refreshing catalog 'omnios' ... Done
 Startup: Caching catalogs ... Done
   Publisher: Using omnios (https://pkg.omniosce.org/bloody/core/).
   Publisher: Using extra.omnios (https://pkg.omniosce.org/bloody/extra/).
       Cache: Using /var/pkg/publisher.
  Installing: Packages (output follows)
Planning: Solver setup ... Done
Planning: Running solver ... Done
Planning: Finding local manifests ... Done
Planning: Fetching manifests:   0/187  0% complete
Planning: Fetching manifests: 173/187  92% complete
Planning: Fetching manifests: 187/187  100% complete
Planning: Package planning ... Done
Planning: Merging actions ... Done
Planning: Checking for conflicting actions ... Done
Planning: Consolidating action changes ... Done
Planning: Evaluating mediators ... Done
Planning: Planning completed in 13.12 seconds
Packages to install: 187
Mediators to change:   2
 Services to change:   5

Download:    0/1263 items  0.0/4.6MB  0% complete
Download: Completed 4.60 MB in 0.10 seconds (0B/s)
 Actions:    1/5322 actions (Installing new actions)
 Actions: Completed 5322 actions in 0.51 seconds.
```

Figure 1. A ZFS file system is created for this zone

This is the basic zone, with most of its properties taking the default values since we chose to omit them. That means this zone has no restrictions on how much memory or CPU it will use.

Thus, we need to define the memory limits.

Define resource limits for a zone

Now that we have a zone, we need to set limits on it. You will mostly be using the following:

- **zone.cpu-cap -** Sets the absolute limit on the amount of CPU resources for this zone can use. You could use a decimal number to express how many CPUs the zone will have available.
- **zone.cpu-shares -** Sets the limit for the number of fair share scheduler (FSS) CPU shares for this zone.
- **zone.max-locked-memory -** Limits the total amount of physical locked memory available to a zone.
- **zone.max-lwps -** Limits the maximum number of LWPs available to this zone, hence enhancing resource isolation.
- **zone.max-swap** - Limits the total amount of swap that can be consumed by user process address space mappings and tmpfs mounts for this zone.

Let's make our zone only able to use 2GB of RAM. First, we need to stop it.

```
# zoneadm -z test halt
```

```
# zonecfg -z test "add rctl;set
name=zone.max-locked-memory;add value
(priv=privileged,limit=2147483648,action=d
eny);end; add rctl;set
name=zone.max-swap;add value
(priv=privileged,limit=2147483648,action=d
eny);end; "
```

If we need to apply this restriction while the zone is running, we need to use **PRCTL(1)**

```
# prctl -n zone.max-swap -r -v 2G `pgrep
-z test init`
# prctl -n zone.max-locked-memory -r -v 2G
`pgrep -z test init`
# prctl -n zone.max-physical-memory -r -v
2G `pgrep -z test init`
```

Tools available to manage zones

The community has created tools to automate the creation of zones. Therefore, you are not forced to use zoneadm or zonecfg to create your zones. Currently, the following tools are under development:

- **LXADM** (http://www.lxadm.org/)

It's a tool created in Perl which allows you to create lx branded zones easily.

- **VMADM** (https://github.com/joyent/smartos-live/)

It's the default tool for managing smart machines (zones) on SmartOS. Additionally, zone definitions are created based on json then feed to vmadm to create the actual zone. You must be running SmartOS to use it.

Conclusion

In this basic walkthrough on zones, we have witnessed the simplicity of use and that the resource control facilities on the operating system allow us to run a considerable number of zones (hardware permitting). We have also not seen lx branded zone which could run unmodified Linux applications.

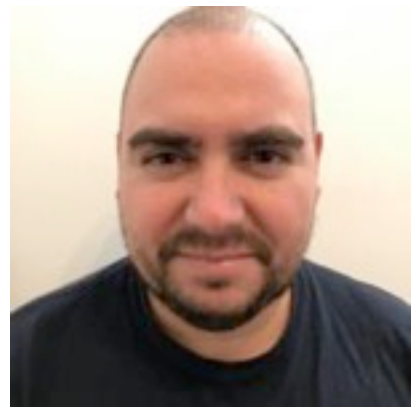
References

https://docs.oracle.com/cd/E36784_01/html/E36848/z.config.ov-13.html

https://omniosce.org/

https://wiki.smartos.org/display/DOC/Building+SmartOS+on+SmartOS

Meet the Author

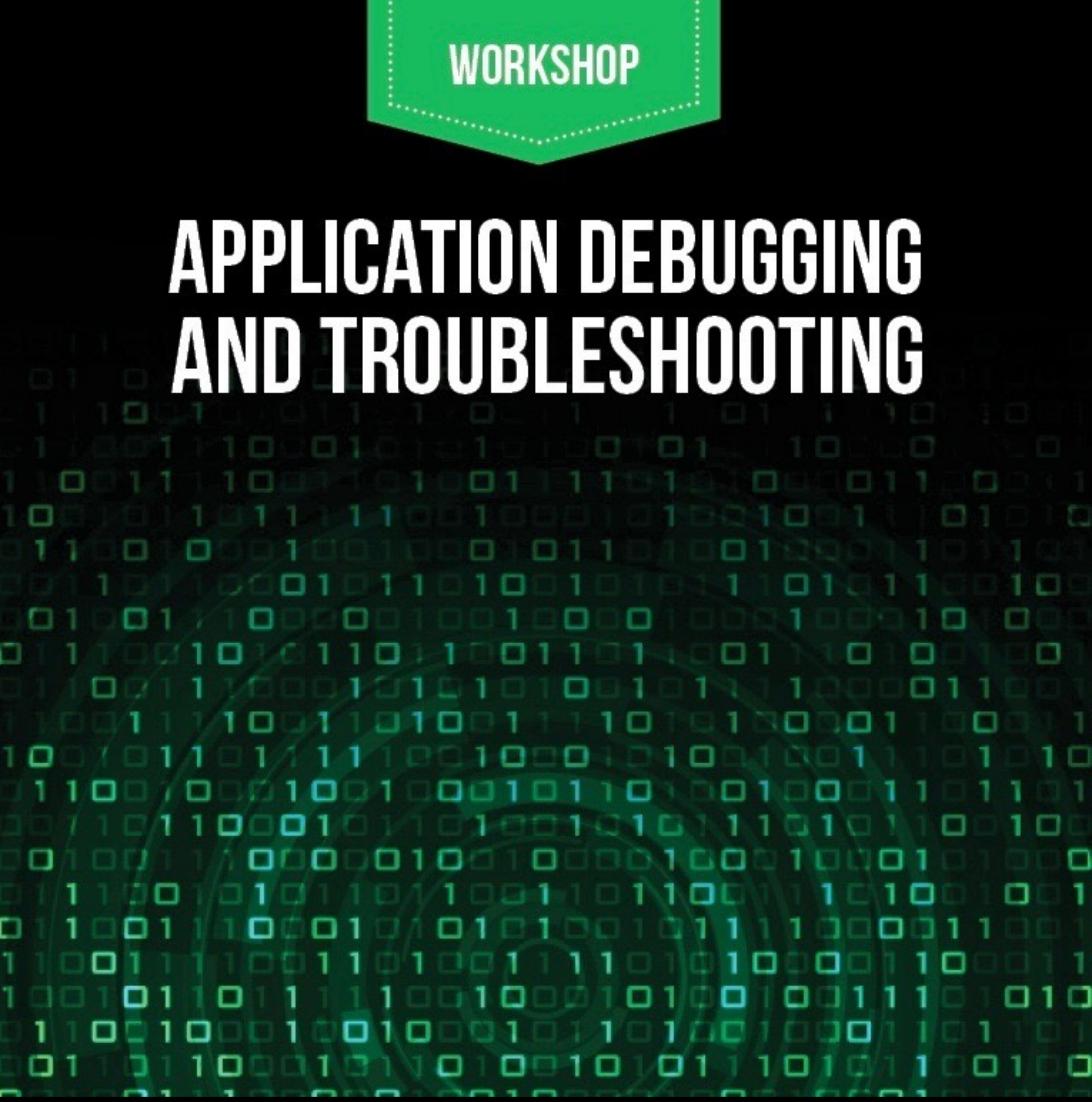


Carlos Neira is a software engineer interested in performance, debuggability, and observability of systems. He has spent most of his career as a software developer debugging issues on Linux, FreeBSD, Solaris, and Z/OS environments.

You can reach him at cneirabustos@gmail.com.

WORKSHOP

APPLICATION DEBUGGING AND TROUBLESHOOTING



Join Us

www.bsdmag.org

bhyvearm64: Virtualization on ARMv8-A

Virtualization is the process of creating a virtual machine that acts like the real hardware for the guest operating system. Efficient virtualization requires hardware features that reduce the overhead usually associated with using virtual machines. Looking to enter the server market, ARM has developed the ARMv8-A architecture which offers such features. We have ported the FreeBSD bhyve hypervisor port to this architecture and we have called the port bhyvearm64.

Introduction

Virtualization has been used since the 1960s and it has gained widespread adoption in recent times due to the benefits offered by virtual machines: partitioning of hardware resources, isolation between virtual machines and the host operating system and encapsulation of an entire system in one file. To enforce the properties that make virtual machines desirable, different software techniques have historically been used: software management of the virtual and physical machine state, shadow page tables, full emulation of interrupts, etc. In some cases, the performance penalty associated with these techniques was quite severe, for example as much as 75% in certain workload scenarios when shadow page tables were used [1].

To get around these limitations hardware designers have implemented different features that reduce the virtualization overhead: Extended Page Tables (EPT, on Intel hardware) or the equivalent Rapid Virtualization Indexing (RVI, on AMD hardware) for memory access, Virtual Machine Control Structure (VMCS) for saving virtual machine state, etc.

bhyvearm64 leverages the features offered by the ARMv8-A architecture to achieve full virtualization on the FreeBSD operating system. The hypervisor is dependent upon the virtualization extensions being implemented by the CPU, which are an optional part of the official architecture specification.

To achieve isolation of the virtual machine and to prevent the guest from directly accessing the hardware, a third CPU execution mode is used, designed specifically for use in a virtualized environment. To reduce the software overhead of managing the guest's paging table structures, a second stage of address translation is used by the hardware to restrict access to the physical memory. And to allow the guest to use I/O, the interrupt controller is partially virtualized in hardware and partially using software emulation.

bhyvearm64 is able to boot a guest FreeBSD operating system and enter user land. However, the virtual machine is limited to a single virtual CPU.

Why ARMv8?

ARM has usually been associated with mobile or low power computing, but in recent times they have been trying to gain a portion of the server market. In a presentation from 2015 they estimate that 20% of the server market will use chips developed by ARM by 2020 [2]. Cavium has recently released version 2 of their ThunderX implementation of the ARMv8-A architecture and in a paper focused on benchmarking Cavium's offering against Intel's [3] the authors note that: „[..] ARM-based processors are now capable of providing levels of performance competitive with state-of-the-art offerings from the incumbent vendors, while significantly improving performance per Dollar".

On the desktop side, Gigabyte is offering ThunderXStation, a development workstation that uses the ThunderX2 chip. The station is aimed at „ARM software development for Android, gaming, embedded and NFV applications" [4].

ARM has also unveiled their Cortex-A76 processor in May 2018 which according to ARM offers „laptop-class performance with mobile efficiency" [5].

Hypervisor architecture

bhyvearm64 has been developed for the ARMv8.0-A version of the 64 bit ARM architecture. The virtualization extensions are required in order for the hypervisor to function. Later revisions of the architecture include several extensions to the virtualization capabilities of the CPU, and in its current state the hypervisor doesn't make use of them.

The arm64 version of the bhyve hypervisor shares the same general architecture with the

x86 version. The hypervisor is composed of three user space utilities that communicate with the hypervisor's vmm kernel module via a special device, as shown in Figure 1.

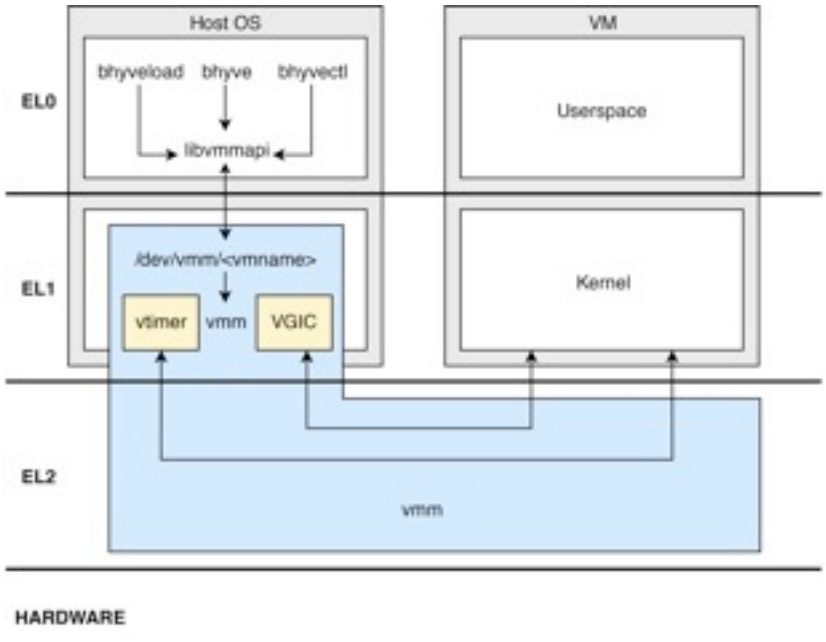


Figure 1. Hypervisor Architecture

Unlike the x86 version of bhyve, the userspace utilities accept only a minimal set of parameters. As the hypervisor evolves, similar parameters will be added.

Listing 1. bhyveload usage

```
Usage: bhyveload [-h] [-k <kernel-image>] [-e
<name=value>] [-b base-address]

                 [-m mem-size] [-l
load-address] <vmname>

       -k: path to guest kernel image

       -e: guest boot environment

       -b: memory base address

       -m: memory size

       -l: kernel load address in the guest
physical memory

       -h: help
```

Listing 1 shows how bhyveload can be used to create a virtual machine with the unique name <vmname>. After the bhyveload process terminates, the virtual machine is created: memory has been allocated, the guest kernel image has been copied to memory, the hypervisor control structures have been initialized and the special device used to communicate with the kernel module is in place. At this point the virtual machine isn't yet running and bhyve can be used to start the guest.

When the FreeBSD arm64 kernel boots on bare metal it expects a pointer to the virtual address where metadata about the kernel image is stored. This metadata contains information about the modules compiled in the kernel and the address and length of the image. The metadata is created by the bootloader by parsing the kernel image. bhyveload parses the guest kernel image in a similar manner to create the list.

Listing 2. bhyve usage

```
Usage: bhyve [-bh] [-c vcpus] [-p pincpu]
<vmname>

       -b: use bvmconsole

       -c: # cpus (default 1)

       -p: pin vcpu 'n' to host cpu 'pincpu +
n'

       -h: help
```

Listing 2 shows how bhyve is used to start the virtual machine. The virtual machine has to be created beforehand by using bhyveload. The <vmname> argument identifies the special device /dev/vmm/<vmname> which will be used to start the guest. A new thread is created for each of the guest CPUs which will wait in an infinite loop for events from the guest, which are used for emulation. At the moment only one thread is started because the hypervisor only supports one virtual CPU per virtual machine.

Of special mention is the -b knob used by bhyve. This means that the guest was compiled to use a special character device, bvmconsole, to print to standard output. bvmconsole guest writes work by emulating MMIO accesses done by the virtual machine and its inner workings will be detailed in the Memory Virtualization section. As for bvmconsole reads, they are done by the guest by polling a special memory location.

bhyvectl is used on x86 to inspect and modify the state of the guest and to shutdown the virtual machine. At the moment we haven't implemented any bhyvectl functionality.

CPU Virtualization

To separate unprivileged processes from the kernel, all ARMv8 CPUs implement two execution modes, or Exception Levels (EL). The first execution mode, EL0, is the least privileged mode used by user processes and EL1 is used for running the FreeBSD arm64 kernel. Any software running in this execution mode has complete control over the hardware.

Because virtualization requires that the guest virtual machine behaves like it is running on bare metal and is otherwise unable to determine that it is executing in a virtualized environment, we have decided to run the guest operating system in the same execution mode – EL1 - as the host kernel. To ensure that the host remains in complete control over the hardware we have used a split hypervisor approach as the ARMv8 version of the KVM hypervisor [6].

As seen in Figure 1, we use a third execution mode, EL2, designed specifically for virtualization. We use a technique called trap-and-emulate for running the virtual machine: a small part of the hypervisor code runs in this execution mode we configure it to trap privileged instructions (instructions that target the hardware directly) executed at lower exception levels.

When the guest is running in EL1, we trap privileged instructions to EL2. Then we change the CPU control configuration to allow direct access to the hardware and we return execution to the host kernel to emulate the instruction on behalf of the guest. When the emulation is done, we resume the guest.

The guest and the host can run on the same CPU and when switching execution between the two we need to have a mechanism for saving and restoring the machine state. On x86, this mechanism is implemented in hardware and each virtual machine has a VMCS structure associated. On ARMv8, this is not available and we had to write the assembly code used for saving and restoring the registers that are changed when a process (in privileged or unprivileged mode) is executing.

Each virtual machine has a struct hyp (a C language structure) associated which represents the hypervisor context for the entire virtual machine. This struct has an array of struct hypctx which represents the virtual CPUs context. At the moment, bhyvearm64 can only manage a virtual machine with a single CPU, but we plan to add support for multiple virtual CPUs in the future.

Memory Virtualization

From the host's perspective, the virtual machine is nothing more than a regular user process, and like all user processes it shouldn't have direct access to physical memory.

One of the first techniques used to restrict guest access to physical memory was shadow page tables. These tables were maintained by the hypervisor and used by the hardware to do address translation. The hypervisor also had access to the guest page tables by trapping writes to the register that holds the physical address of the tables. Each page fault was then intercepted by the hypervisor which allocated physical frames as needed by the guest [7].

Hardware designers have implemented a technique to restrict a virtual machine's access

of the physical memory similar to how the kernel treats user processes. When it comes to regular processes (that is, not a virtual machine), the kernel uses page tables which translate a program's virtual address into a physical address. For virtual machines the approach is similar: all the addresses generated by the guest as a result of address translation are subjected to a second translation step which results in the real physical address.

This technique has different names when implemented by different manufacturers: Extended Page Tables (EPT) on Intel hardware, Rapid Virtualization Technology (RVI) on AMD CPUs, and Stage 2 Translation on ARMv8 CPUs.

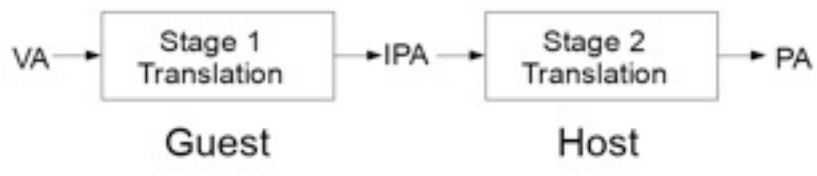


Figure 2. Stage 2 Translation

Figure 2 is a schematic of this process. The guest maintains its own page tables, just like the host kernel does. Remember that the guest must not be aware that it is running in a virtualized environment. When the guest is running, we activate the second stage of translation and all the physical addresses generated by the guest (ARM calls these addresses Intermediate Physical Addresses, or IPA) will be translated using a different set of page tables to the real physical address. The second set of page tables is created and maintained by the hypervisor and this is how the guest has restricted memory access.

Using the second stage of translation also provides the added benefit of being able to control when we want to trap guest memory accesses for emulation purposes. An exception is created when the virtual machine tries to read or write a guest's physical address that isn't mapped in the stage 2 tables. The hypervisor then switches execution to the host and emulation can be performed. This technique is used when the guest tries to write to standard output by using bvmconsole and emulating the memory-mapped components of the interrupt controller.

The second stage translation tables have a different format than the page tables used by the kernel. We have implemented the second stage tables in the machine-dependent part of the FreeBSD memory subsystem.

Interrupt Virtualization

Input/output devices are significantly slower than the CPU. To communicate efficiently with these devices, interrupts are used. Interrupts are electrical signals that are asynchronous (they can come at any time regardless of what the CPU is doing) and when they become active, the CPU starts executing code from a predefined address from memory which depends on the type of interrupt.

A virtual machine isn't complete without virtual interrupts. At the bare minimum, the guest OS needs timer interrupts to be able to keep track of time and perform process scheduling.

We have focused on virtualizing version 3 of the ARM interrupt controller, called Generic Interrupt Controller v3, or GICv3.

The GIC uses a mixture of system registers and memory-mapped registers to configure and control interrupts. Figure 3 represents the three main components of the interrupt controller.

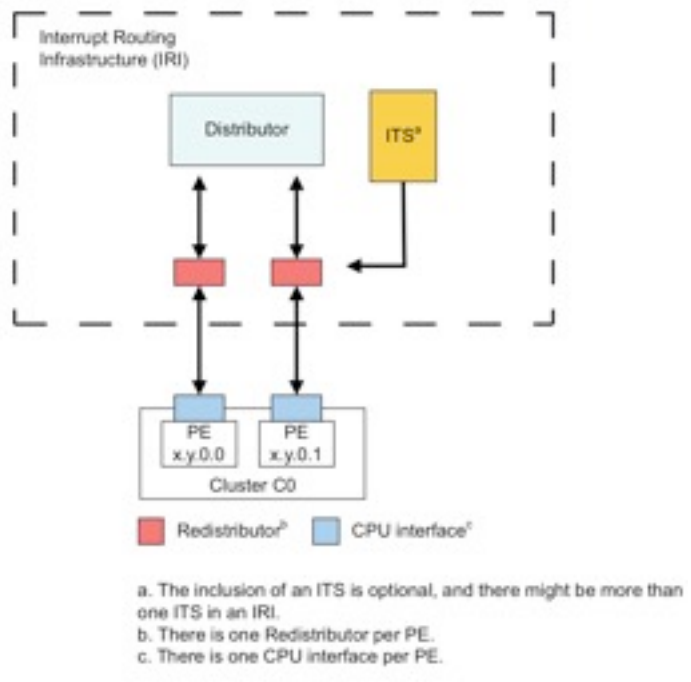


Figure 3. GIC Architecture [8]

The Distributor and Redistributor are memory-mapped and the CPU interface can be accessed by using physical registers. The Distributor controls global interrupts, called Shared Peripheral Interrupts (SPI), and the Redistributor controls interrupts targeted at one CPU: the Software Generated Interrupts (SGI) and Private Peripheral Interrupts (PPI). The CPU interface is used for processing interrupts.

The fourth type of interrupts are the Locality-specific Peripheral Interrupts (LPI) which are message-based interrupts. They are controlled by the Interrupt Translation Service (ITS in the image). We haven't virtualized LPIs or the ITS.

Virtualization of the interrupt controller is done in part in hardware and in part in software. For the CPU interface registers, the GIC provides hardware-assisted virtualization which is controlled by software running in EL2. For memory-mapped components, the virtualization is done by using trap-and-emulate instead. The hypervisor stores the state of the Distributor and Redistributors and each memory access to these registers is trapped. For each trapped instruction the hypervisor updates the state of the emulated registers accordingly.

Timer Virtualization

A timer is essential for any operating system: without it, there could be no process scheduling. Operating systems also use a timer for scheduling periodic tasks, either as a functionality offered to user space processes, or for internal use.

The timer on the ARMv8 platform is called the Generic Timer. The Generic Timer is composed of two different components: the physical timer and the virtual timer, which shows the same time as the physical timer minus a fixed offset.

For virtualizing the timer, the ARMv8.0 platform provides controls to trap access to the physical timer (later extensions provide controls to trap accesses to the virtual timer too). When a timer is scheduled to fire by the guest, we create a callout in the host which will inject the corresponding interrupt.

Conclusion

Using hardware-assisted virtualization we have been able to write a working hypervisor on the ARMv8-A platform. The virtualization features offered by ARM serve a similar purpose as those available on the x86 platform.

The architecture provides a distinct privilege level which we have used to control a guest to provide the isolation expected from a virtual machine.

To restrict the guest's access to physical memory we have used a second address translation step called Stage 2 Translation. This has required some changes to the machine-dependent pmap component of the FreeBSD memory subsystem.

Interrupt controller virtualization was the first step in making the virtualization of I/O devices possible. Using this virtualized interrupt controller we have then created a timer for the virtual machine.

References

Advanced Micro Devices, Inc. (2018, Jun.) *AMD-V Nested Paging.* [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf

The Register (2018, Jun.) *ARM plans to win 20 per cent of the server market by the year 2020.* [Online]. Available: https://www.theregister.co.uk/2015/03/23/arm_plans_to_win_20_per_cent_of_the_server_market_by_the_year_2020/

S. McIntosh-Smith, J. Price, T. Deakin and A. Poenaru (2018, Jun.) *Comparative Benchmarking of the First Generation of HPC-Optimised Arm Processors on Isambard.* [Online]. Available: https://uob-hpc.github.io/assets/cug-2018.pdf

Cavium Inc. (2018, Jun.) *Gigabyte announces ThunderXStation: Industry's first Armv8 Workstation based on Cavium's ThunderX2 Processor*. [Online]. Available: https://cavium.com/news/gigabyte-announces-thunderxstation-industry-s-first-armv8-workstation-based-on-cavium-s-thunderx2-processor

Anandtech (2018, Jun.) *Arm's Cortex-A76 CPU Unveiled: Taking Aim at the Top for 7nm*. [Online]. Available: https://www.anandtech.com/show/12785/arm-cortex-a76-cpu-unveiled-7nm-powerhouse

K. Dall and J. Nieh (2018, Jun.) *KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor.* [Online]. Available: http://www.cs.columbia.edu/~nieh/pubs/asplos2014_kvmarm.pdf

J. E. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, Morgan Kaufmann Publishers, 2005.

ARM Limited. (2018, Jun.) ARM Generic Interrupt Controller Architecture Specification GIC architecture version 3.0 and version 4.0. [Online]. Available: https://static.docs.arm.com/ihi0069/d/IHI0069D_gic_architecture_specification.pdf

Links

bhyvearm64 project: https://github.com/FreeBSD-UPB/freebsd/tree/projects/bhyvearm64

Utilities and tutorial for installing and running bhyvearm64: https://github.com/FreeBSD-UPB/bhyvearm64-utils

Meet the Author

Alexandru Elisei is a 4th year college student studying Computer Science at University Politehnica of Bucharest. He is very passionate about computers and open source software. He has made contributions to various open source projects, like Gentoo or Moodle, as well as taking part in the Google Summer of Code program as a student developer.

MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

IMPROVE YOUR POSTGRESQL SKILLS

WORKSHOP

BY

LUCA FERRARI

VOL 12 NO 07
ISSUE 07/2018 (107)
ISSN 1898-9144

FreeBSD

iSCSI On FreeBSD

- *What is iSCSI?*
- *File-Level Or Block-Level?*
- *iSCSI and ZFS*
- *FreeBSD iSCSI Target*
- *iSCSI Mutual Authentication*
- *Windows 2016 as iSCSI Initiator*
- *Windows 2016 iSCSI Tuning*

What is iSCSI?

iSCSI is a protocol that gives you the ability to share storage over a network at block level. It’s like connecting new storage to your computer and can format it as you wish.

In iSCSI terminology, the computer that shares the storage is known as the *target,* and the clients which access the iSCSI storage are called *initiators*.

FreeBSD originally supports kernel-based iSCSI target and initiator.

File-Level Or Block-Level?

It's really up to you. Many people are not quite sure about choosing between DAS(Block-Level directly), NAS(File-Level over the network) and SAN(Block-Level over the network). Don’t settle for a storage based on the amount of space only, rather, the answers to these important questions should act as a guiding principle

What is your storage expansion policy?

If you can expand your storage locally and have a linear expansion ratio, it means you have suitable time and resources to prepare your storage. Therefore, you can use DAS, NAS, SAN or combine them as you want. But if you can't estimate the growth ratio and it's not linear, it's better to choose something over the network, like NAS or SAN.

What is your backup policy?

There are three types of backup: Full, Incremental, and Differential.

Incremental backs up only the changed data since the last full or incremental backup, whereas the differential backs up only the changed data since the last full backup. Incremental backup is most suitable for network-enabled like NAS or SAN because of the needed network bandwidth.

What is your access policy?

If you have to write at the same time in the same area, NAS is required because block-level access can corrupt your data.

iSCSI and ZFS

ZFS is capable of creating a volume of the given size as a block device in /dev/zvol/path, where the path is the name of the volume in the ZFS namespace. Then, iSCSI can use this block device like a separate hard disk.

The point is, zvol must be fixed size and can't expand later (you can't change your hard disk size). However, you can use multiple zvols at RAID manner on your initiator and must stop writing on backup (incremental backup) within short moments.

FreeBSD iSCSI Target

FreeBSD manages the iSCSI with a configuration file located in /etc/ctl.conf. Add a line to /etc/rc.conf to make sure the ctld daemon is automatically started at boot, and then start the daemon.

```
# sysrc ctld_enable=YES
```

Below is a sample of ctl.conf :

```
portal-group pg0 {

        discovery-auth-group
no-authentication

        listen 192.168.1.10

}



portal-group pg1 {

        discovery-auth-group
no-authentication

        listen 192.168.2.10

}
```

```
auth-group ag0 {

        chap iscsi1 iscsi0pass123456

}


auth-group ag1 {

        chap iscsi2 iscsi1pass123456

}



target
iqn.2018-05.com.meetbsd.storage:target0 {

        auth-group ag0

        portal-group pg0

        lun 0 {

                path
/dev/zvol/storage/iscsi_0

                size 10G

        }

}



target
iqn.2018-05.com.meetbsd.storage:target1 {

        auth-group ag1

        portal-group pg1

        lun 1 {

                path
/dev/zvol/storage/iscsi_1

                size 10G

        }

}
```

This config file mainly includes three sections:

Portal-groups

It contains network setting like discovery, listening IP and port.

Auth-group

It contains authentication method, user, and password.

Target

It contains portal-group, auth-group and LUN(logical unit number).

LUN defines path and size of allocation plus other options.

Since we have two interfaces with 192.168.1.10 and 192.168.2.10 IP addresses and want use them simultaneously, we need to create a two portal-group that requires no password to discover on the client side.

Then, let's create a two auth-group with username and password. This authentication method is CHAP (Challenge-Handshake Authentication Protocol). CHAP means the password never in use directly, instead, both client and server use a challenge message and one-way hash to verify authentication.

This modular config file lets you separate network aspects from another, and you can manage efficiently.

It's worth noting that the password must be eight digits at least.

Then, start ctld by:

```
# service ctld start
```

iSCSI target will listen on port 2360, and everything will be in order. But if you later change this config file, then issue the following command:

```
# service ctld reload
```

iSCSI Mutual Authentication

iSCSI supports two types of authentication:

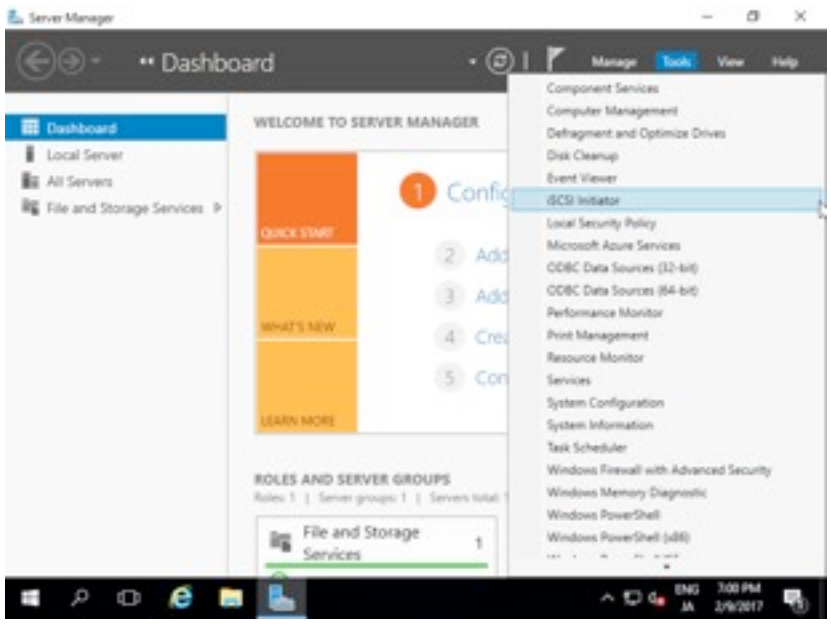
One-Way CHAP (or simply CHAP), the target only authenticates the initiator.

Mutual CHAP, the initiator also authenticates the target.

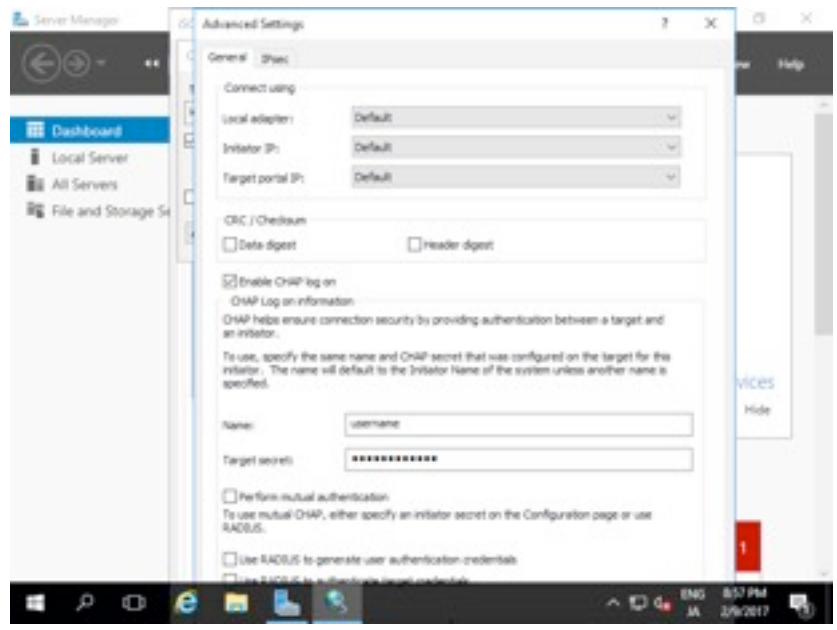
Mutual authentication, also called two-way authentication, is a process in which the client authenticates the server and vice-versa. If some pretend to be your storage, all of your valuable data will easily be compromised.

Windows 2016 as iSCSI Initiator

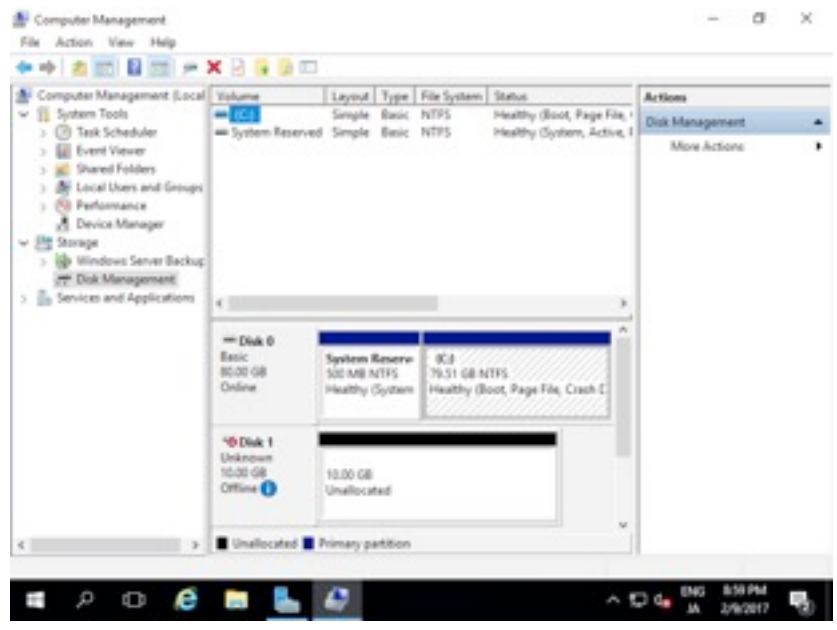
Windows 2016 supports iSCSI and can act as the iSCSI initiator.



The only items you have to set are the IP of the iSCSI target (server), your username and password.



hen you must format the iSCSI disk and create a partition.



Windows 2016 iSCSI Tuning

The only proven tuning for iSCSI on Windows 2016 that improves performance by about 13% is Jumbo Frames.

A larger MTU (maximum transmission unit) typically decreases the amount of CPU utilization.

The Jumbo Frame can carry up to 9000 byte MTUs, which leads to better iSCSI performance.

Here is how to enable Jumbo Frame on interface name iSCSI:

First press Window+R and then issue the following command:

```
# netsh interface ipv4 set sub interface
"iSCSI" mtu=9000 store=persistent
```

Conclusion

Many people are not sure about choosing between DAS (Block-Level directly), NAS (File-Level over the network) and SAN (Block-Level over the network). Don't settle for a storage based on the amount of space only, rather, the answers to these important questions should act as a guiding principle:

- What is your storage expansion policy?
- What is your backup policy?

Useful Links

https://www.freebsd.org/doc/handbook/network-iscsi.html

https://www.server-world.info/en/note?os=Windows_Server_2016&p=iscsi&f=3

Meet the Author

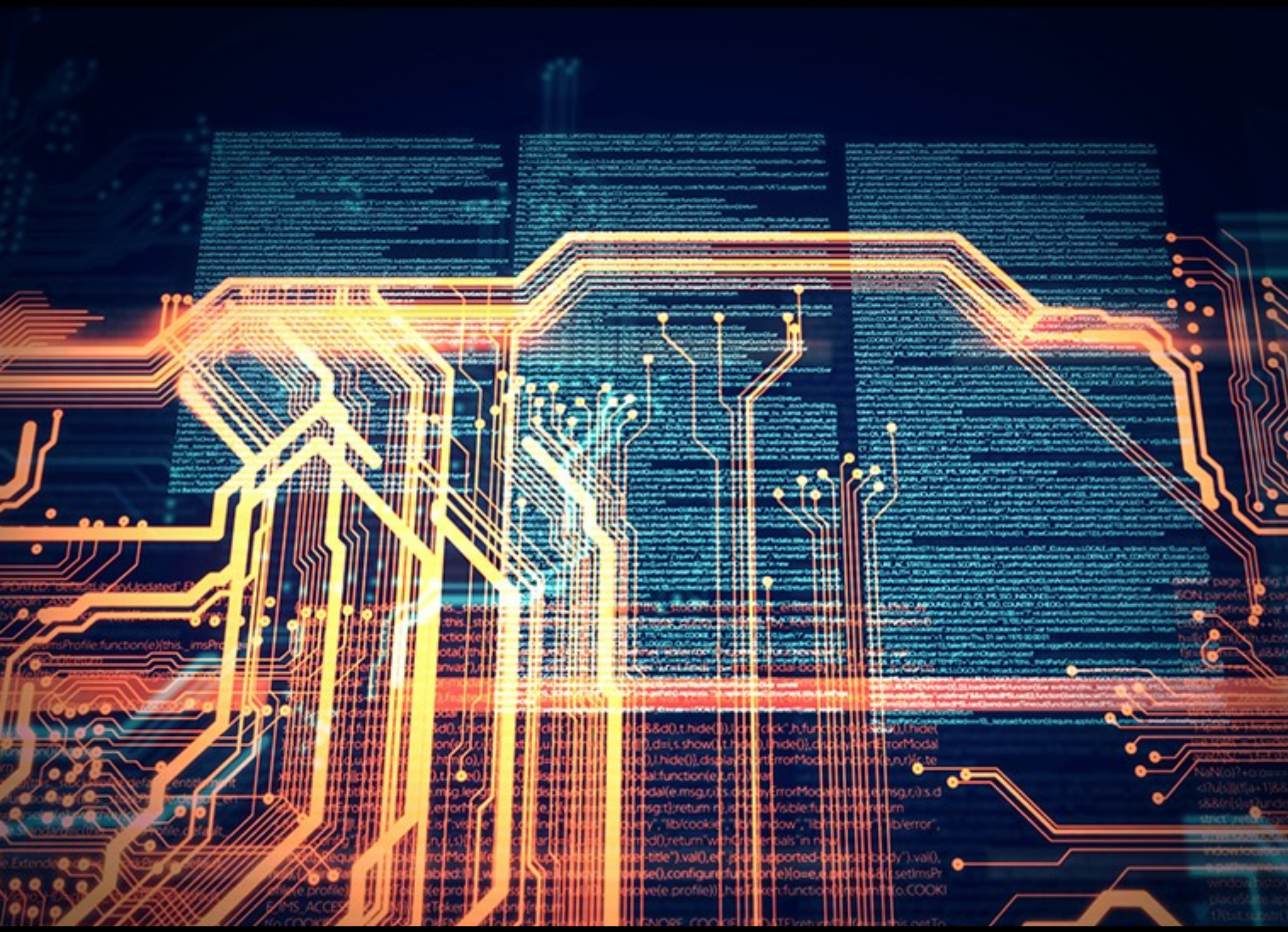
Abdorrahman Homaei has been working as a software developer since 2000 and has used FreeBSD for more than ten years. He became involved with the meetBSD dot ir and performed serious trainings on FreeBSD. He started his company (etesal amne sara Tehran) in Feb 2017 and it is based in Iran Silicon Valley.

Full CV: ***http://in4bsd.com***

His company: ***http://corebox.ir***

WORKSHOP

Device Driver Development for BSD



Join Us

www.bsdmag.org

HTTP/2 and PHP with Apache on FreeBSD: Not as Simple as It Seems

In an earlier article, I showed you how to run FreeBSD on Google Compute Engine, running an Apache web server with PHP. Now let's see how to improve its performance with the latest version of HTTP.

HTTP/2 has significant advantages over earlier versions. However, it and PHP don't work together "out of the box" on FreeBSD, and what appears to be the appropriate fix breaks an otherwise functioning web server.

Follow my investigation of the mystery, and at the end, I'll have assembled a working configuration for you.

Why HTTP/2?

HTTP/2 provides better performance than HTTP/1.1. The HTTP headers are compressed, and multiple HTTP requests are sent through a single TCP connection (called *HTTP pipelining* or *multiplexing*).

Recall that each TCP connection requires a 3-way handshake to set it up, and another 3-way handshake to shut it down. Moving everything through one TCP connection saves a lot of back-and-forth exchanges to set up and tear down multiple connections. Mobile clients have greater latency, and all those additional TCP connections can slow things down.

Concurrency - using one TCP pipeline for several sets of data - can be especially helpful with mobile clients. Some hosting providers claim that HTTP/2 can cut page load times in half.

Other HTTP/2 features can provide further performance advantages *if* the server and client can take advantage of them. If the server can anticipate which additional resources are needed for a given page, *server push* is the technique of "pushing" those resources to the client before the client analyzes the initial content and realizes that it needs to request them. See the Apache documentation for details on server push.

From the opposite direction, *stream prioritization* is a technique by which the client can prioritize certain data streams over others. It can request components like images and CSS and JavaScript files in an order that speeds page rendering.

Site speed is a Google search ranking factor. So, while they don't specifically rank on support for HTTP/2, its speed improvements will help your ranking.

How Did We Get HTTP/2?

HTTP appeared, received an initial tweak, and then stayed the same for almost two decades. Its design was about as stable as boring old UDP. Meanwhile, HTTP became the Internet's most-used application protocol. Then Google started stirring things up in its quest for web performance.

HTTP was initially developed in the early 1990s. Remember Netscape Navigator and NCSA's Mosaic before that? You very likely don't, but that's what they were designed for. Version 1.0 was formally defined in 1996 and version 1.1 followed in early 1997. Then things were quiet for a decade, and it took almost 20 years before HTTP/2 arrived.

Google designed the SPDY protocol in the early 2010s. It aimed to increase page load speed and was supported by several browsers.

SPDY evolved into HTTP/2, which was published as a standard in 2015. Most browsers have supported HTTP/2 since 2015.

Your server will need support from its shared libraries. You need a TLS library that supports the ALPN protocol. That means at least OpenSSL 1.0.2 (Jan 2015), LibreSSL 2.1.3 (Jan 2015), or GnuTLS 3.2.0 (May 2013). That's no problem on recent FreeBSD, but someone locked into older Red Hat or CentOS Linux distributions may be unable to run HTTP/2. On FreeBSD, try the following commands. The first one checks the standard command included in the base operating system. The second checks the version installed when you add the libressl or openssl package. The openssl package provides a slightly newer version of the binary and shared libraries, plus a large collection of manual pages.

```
$ openssl version

$ /usr/local/bin/openssl version

$ gnutls-cli --version

$ pkg info | egrep
'openssl|gnutls|libressl'
```

HTTP/2 is an *alternative* to HTTP/1.1 and HTTP/1.0, not a replacement. It includes a negotiation mechanism, so the client and server can use HTTP/1.0, HTTP/1.1, or HTTP/2.

You will find entries for HTTP/1.0 clients in your log, but you will probably find that most are automated indexing bots. I was surprised to learn that many indexing bots run the original

protocol. However, I suppose that HTTP/2 doesn't have a big advantage in that specific situation.

Getting Started

In a previous article, I described how I had deployed a FreeBSD server in the Google Cloud. (see "FreeBSD, Google Cloud, and Dual ECC/RSA", *BSD Magazine,* November 2017, https://bsdmag.org/download/openldap-directory-services-freebsd/)

The server seemed to be working fine, and as far as serving out the appropriate data, it was. I didn't notice at first that it wasn't supporting HTTP/2. Then I read an article listing recent estimates of the percentage of clients using HTTP/2. Being curious, I used some simple grep -c commands to count how many lines in the Apache log file contain the strings "HTTP/1.0", "HTTP/1.1", and "HTTP/2". I was surprised to find no entries for HTTP/2!

After tracking down what seemed to be missing, I made what seemed to be the appropriate changes. But now Apache would not start! I fixed that and then broke something else. After some investigation, I got everything working. Here's what I discovered:

The sample httpd.conf **file supplied in the FreeBSD Apache24 package does not support HTTP/2.** It fails to do so because of an error that **is not logged** using that configuration file. Then, what seems to be the obvious solution prevents Apache from starting. Apache configuration debugging doesn't make for a thrilling tale, but there *were* mysteries to solve.

Follow along with my experiments and debugging. By the time we reach the end of the article, I will have it all working!

Enabling HTTP/2 and PHP with Apache Modules

I had first set out to enable HTTP/2 and PHP. I need to treat all HTML files as PHP, as I use PHP to insert a standard footer on every page on the site, among other tasks. I had added the following to my other changes near the end of the httpd.conf file:

```
[... many lines not shown ...]



# Enable HTTP/2

LoadModule http2_module
libexec/apache24/mod_http2.so

# Prefer HTTP/2 over TLS, then HTTP/2
without TLS, then HTTP/1.1, finally
HTTP/1.0

Protocols h2 h2c http/1.1



# Set up PHP

# NOTE: This line will be replaced by the
time we're done,

# so keep on reading to see what really
goes here...

LoadModule php7_module
libexec/apache24/libphp7.so



[... many lines not shown ...]
```

Mysterious Failure to Support HTTP/2

I restarted Apache and used curl to request just the header (with -I), ignoring the fact that the certificate is not suitable for host name localhost (with -k), over HTTP/2 (with --http2). But I did not get what I expected:

```
$ curl -I -k --http2 https://localhost/

HTTP/1.1 200 OK
Date: Wed, 30 May 2018 13:51:06 +0000
```

```
Server: Apache/2.4.29 (FreeBSD)
OpenSSL/1.0.2k-freebsd PHP/7.1.14

Upgrade: h2,h2c

Connection: Upgrade

Last-Modified: Tue, 22 May 2018 09:21:52
+0000

ETag: "fc2-55dbe74d77540"

Accept-Ranges: bytes

Content-Length: 4034

Content-Type: text/html charset=UTF-8
```

I verified that I had loaded the mod_http2.so module, and that the server reported no error or warning when restarting. **There** *was* **an error** but the server did not log it by default. Here is all I got:

```
# grep mod_http2.so
/usr/local/etc/apache24/httpd.conf

LoadModule http2 module
libexec/apache24/mod_http2.so

# tail /var/www/logs/httpd-error.log

[...]

[Thu May 31 10:02:42.43981 2018]
[mpm_prefork:notice] [pid 6201] AH00163:
Apache/2.4.29 (FreeBSD)
OpenSSL/1.0.2k-freebsd PHP/7.1.14
configured -- resuming normal operations

[Thu May 31 10:02:42.43981] [core:notice]
[pid 6201] AH00094: Command line:
'/usr/local/sbin/httpd -D NOHTTPACCEPT'
```

This took a while to track down. Eventually I added a directive to httpd.conf for some logging from the http2 module:

```
[... many lines not shown ...]


# Enable HTTP/2

LoadModule http2_module
libexec/apache24/mod_http2.so

Protocols h2 h2c http/1.1

<IfModule http2_module>

        LogLevel http2:info

</IfModule>


# Set up PHP

LoadModule php7_module
libexec/apache24/libphp7.so


[... many lines not shown ...]
```

Then I restarted the server and looked at the end of the error log. **There had been a problem, but without the elevated logging Apache did not report it.** See the error code AH10034 below.

```
# /usr/local/etc/rc.d/apache24 restart

Performing sanity check on apache24
configuration:

Syntax OK

Stopping apache24.

Waiting for PIDS: 7138.

Performing sanity check on apache24
configuration:

Syntax OK

Starting apache24.

# tail /var/www/logs/httpd-error.log

[...]

[Thu May 31 10:47:33.000130 2018]
[mpm_prefork:notice] [pid 6956] AH00169:
caught SIGTERM, shutting down

[Thu May 31 10:47:33.002498 2018]
[http2:info] [pid 7138] AH03090: mod_http2
(v1.10.12,
```

```
feats=CHPRIO+SHA256+INVHD+DWINS, nghttp2
1.29.0), initializing...

[Thu May 31 10:47:33.011375 2018]
[http2:warn] [pid 7138] AH10034: The mpm
module (prefork.c) is not supported by
mod_http2. The mpm determines how things
are processed in your server. HTTP/2 has
more demands in this regard and the
currently selected mpm will just not do.
This is an advisory warning. Your server
will continue to work, but the HTTP/2
protocol will be inactive.

[Thu May 31 10:47:33.217357]
[mpm_prefork:notice] [pid 7138] AH00163:
Apache/2.4.29 (FreeBSD)
OpenSSL/1.0.2k-freebsd PHP/7.1.14
configured -- resuming normal operations

[Thu May 31 10:47:33.293589] [core:notice]
[pid 7138] AH00094: Command line:
'/usr/local/sbin/httpd -D NOHTTPACCEPT'
```

Apache MultiProcessing Modules

Apache provides a variety of MPMs or MultiProcessing Modules. In general, one master or mother process running as root is started by the apache24 script. The privileged mother process opens the privileged TCP ports 80 and 443. It then starts a number of child processes. The child processes inherit the open ports and then call setuid() to change their user identity to the relatively unprivileged www user. Then they do the actual work of serving out data.

There are multiple MPMs to choose from. They work in slightly different ways and offer different advantages.

I had started with a copy of the default httpd.conf.sample and so the server was using the mod_mpm_prefork.so module. But that is not compatible with the mod_http2.so module. The server starts and provides all functionality, *except* it will not use HTTP/2.

The prefork module implements a non-threaded pre-forking server. A single control process launches child processes which listen for connections and serve them. It tries to always maintain a few idle server processes, so clients do not need to wait for a new child server process to be forked.

The worker module implements a hybrid multi-process multi-threaded server. It can serve a large number of requests while using fewer system resources than the prefork module would require. A single control process launches child processes. Each child process creates a fixed number of server threads, plus a listener thread that listens for connections and passes each one to a server thread. Again, it always tries to maintain a pool of idle server threads, so clients do not need to wait for new threads or processes to be created.

The event **module** was based on the worker module, and it was created for Apache 2.4 because Apache 2.2 was significantly slower than Nginx. It uses several processes and several threads per process in an asynchronous event-based loop. This gives performance equal to or slightly better than event-based web servers.

Yes, people are working on other experimental MPMs including Threadpool, Leader, and Perchild. But with stock Apache 2.4, your choices are prefork, worker, and event.

I enabled the mod_mpm_event.so module, as that should provide the best performance. Here's what I put in httpd.conf:

```
[... many lines not shown ...]

LoadModule mpm_event_module
libexec/apache24/mod_mpm_event.so

#LoadModule mpm_prefork_module
libexec/apache24/mod_mpm_prefork.so

#LoadModule mpm_worker_module
libexec/apache24/mod_mpm_worker.so

[... many lines not shown ...]
```

Now Apache would not start, because the mod_php module is not compiled to be thread-safe:

```
# /usr/local/etc/rc.d/apache24 restart

Performing sanity check on apache24
configuration:

[Thu May 31 10:56:50.293589 2018]
[php7:crit] [pid 7604:tid 34397577216]
Apache is running a threaded MPM, but your
PHP Module is not compiled to be
threadsafe.  You need to recompile PHP.

AH00013: Pre-configuration failed
```

I wanted the higher performance of HTTP/2 and multi-threaded web server processes, but I did *not* want the added maintenance work of compiling and installing my own PHP module. I wanted to use the OS packages so that I would get PHP module updates automatically when they became available.

The solution

I want to use the event multiprocessing module, so I commented out the line loading libphp7.so.

I added package php72. It includes php-fpm, the PHP FastCGI Process Manager. It runs as a daemon, listening on a TCP socket for CGI requests (TCP port 9000 on localhost only by default).

Next, I added a line to /etc/rc.conf

```
php_fpm_enable=YES
```

I started php-fpm and verified that it is listening. It uses the same model of starting as root and spawning multiple unprivileged workers.

```
# /usr/local/etc/rc.d/php-fpm start

Performing sanity check on php-fpm
configuration:

[14-Feb-2018 15:59:00] NOTICE:
configuration file
/usr/local/etc/php-fpm.conf test is
successful


Starting php_fpm.

# lsof  -i | egrep 'PID|php'

COMMAND   PID USER   FD   TYPE
DEVICE SIZE/OFF NODE NAME

php-fpm 40778  www    0u  IPv4
0xfffff800100f9410      0t0  TCP
localhost:9000 (LISTEN)

php-fpm 63636  www    0u  IPv4
0xfffff800100f9410      0t0  TCP
localhost:9000 (LISTEN)

php-fpm 64754  www    0u  IPv4
0xfffff800100f9410      0t0  TCP
localhost:9000 (LISTEN)

php-fpm 91117 root    7u  IPv4
0xfffff800100f9410      0t0  TCP
localhost:9000 (LISTEN)
```

I need to make a few more changes, telling the server to use Index.html as the default file for a directory, and to handle all HTML files with PHP.

Use Index.html (and not index.* or *.htm)

An *index file* is used when the client requests a directory. For example, http://cromwell-intl.com/ will be treated as a request for the index file in the root directory of the website. This requires the mod_dir module. Apache should load that module by default, but let's check.

Index.html is my site's standard index file name, versus Apache's default lower-case index.html. Here's what I changed in httpd.conf.

```
[... many lines not shown ...]

LoadModule dir_module
libexec/apache24/mod_dir.so

[... many lines not shown ...]
```

```
# DirectoryIndex: sets the file that
Apache will serve if a directory

# is requested.

#

<IfModule dir_module>

    ## DirectoryIndex index.html

    DirectoryIndex Index.html

</IfModule>


[... many lines not shown ...]
```

Treat All HTML Files As PHP

All my pages use PHP to include standard headers and footers, to create microdata for search engine and social media indexing, and to load Google AdSense code blocks. Every page needs PHP. However, I have always named the files "*.html". **So, all HTML files must be treated as PHP.**

This is a UNIX-family operating system, where filename extensions don't matter. Until, of course, they do. The php-fpm daemon cares about file name extensions!

Edit /usr/local/etc/php-fpm.d/www.conf to tell it to accept files named both *.php *and* *.html. Without that, passing it a file named *.html results in a very simple page saying "Access denied".

```
[... many lines not shown ...]


; Limits the extensions of the main script
FPM will allow to parse. This can

; prevent configuration mistakes on the
web server side. You should only limit

; FPM to .php extensions to prevent
malicious users to use other extensions to

; execute php code.

; Note: set an empty value to allow all
extensions.

; Default Value: .php

;security.limit_extensions = .php .php3
.php4 .php5 .php7

security.limit_extensions = .php .html


[... many lines not shown ...]
```

Now I can tell Apache to pass all files named *.html to the PHP proxy. Putting all of this together, here are the changes and additions to httpd.conf:

```
[... many lines not shown ...]

LoadModule dir_module
libexec/apache24/mod_dir.so

[... many lines not shown ...]


LoadModule mpm_event_module
libexec/apache24/mod_mpm_event.so

#LoadModule mpm_prefork_module
libexec/apache24/mod_mpm_prefork.so

#LoadModule mpm_worker_module
libexec/apache24/mod_mpm_worker.so


[... many lines not shown ...]


# DirectoryIndex: sets the file that
Apache will serve if a directory

# is requested.

#

<IfModule dir_module>
```

```
    ## DirectoryIndex index.html

    DirectoryIndex Index.html

</IfModule>


[... many lines not shown ...]


# Enable HTTP/2

LoadModule http2_module
libexec/apache24/mod_http2.so

# Prefer HTTP/2 over TLS, then HTTP/2
without TLS, then HTTP/1.1, and finally
HTTP/1.0

Protocols h2 h2c http/1.1

<IfModule http2_module>

        LogLevel http2:info

</IfModule>


# Set up PHP

LoadModule proxy_module
libexec/apache24/mod_proxy.so

LoadModule proxy_http2_module
libexec/apache24/mod_proxy_http2.so

LoadModule proxy_fcgi_module
libexec/apache24/mod_proxy_fcgi.so

<FilesMatch \.html$>

        SetHandler
"proxy:fcgi://127.0.0.1:9000"

</FilesMatch>


[... many lines not shown ...]
```

The above will work as long as the clients don't ask for files that don't exist. In that case, the client will get a simple "File not found" page, and a "Primary script unknown" entry appears in the error log. Add the following to the end of your .htaccess file to solve that problem. Don't duplicate the RewriteEngine line if it's already in the file:

```
# Do NOT duplicate the following line if
it

# already exists earlier in the file.

RewriteEngine on


# If the client asks for a specific
non-existent *.html file, rewrite it

# so it isn't passed to the PHP engine
causing a "Primary script unknown"

# log entry and a plain "File not found"
page.

RewriteCond %{REQUEST_FILENAME} \.html$

RewriteCond
%{DOCUMENT_ROOT}/%{REQUEST_URI} !-f

RewriteRule (.*) - [H=text/html]
```

Restart apache24 and test it. This needs to include testing with requests for nonexistent directories and files.

You will find suggestions to use ProxyPassMatch instead of SetHandler in the above use of the PHP proxy. That does not work with .htaccess redirection, and you'll get a simple "File not found" error for cases where it should have been redirected.

Now my PHP inclusion works, so all my pages get the standard header with automatically generated social media microdata, the standard footer, and the Google AdSense ads. The pages now look as they should! And, they should be served more efficiently with HTTP/2.

Testing HTTP/2

Let's make sure that HTTP/2 is working:

```
$ curl -I -k --http2
https://cromwell-intl.com/

HTTP/2 200 OK
Date: Thu, 31 May 2018 12:11:06 +0000

Server: Apache/2.4.29 (FreeBSD)
OpenSSL/1.0.2k-freebsd PHP/7.1.14

Upgrade: h2,h2c

Connection: Upgrade

Last-Modified: Tue, 22 May 2018 09:21:52
+0000

ETag: "fc2-55dbe74d77540"

Accept-Ranges: bytes

Content-Length: 4034

Content-Type: text/html charset=UTF-8
```

That looks good!

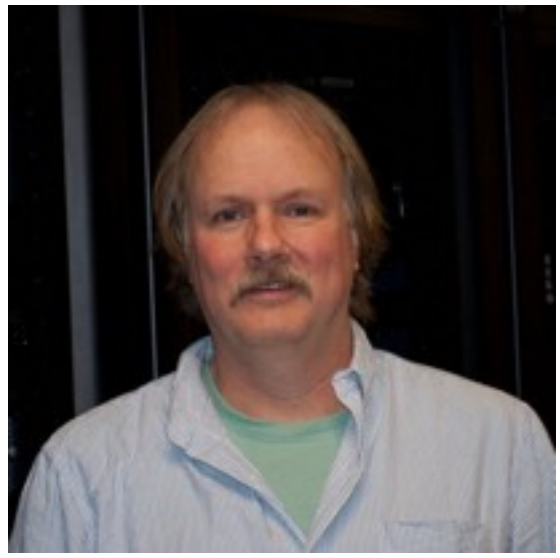
Final Cleanup

Be careful! If you have installed the mod_php71 package, then a package update will run a post-installation script that re-inserts the troublesome libphp7.so line in your configuration file automatically!

Remove the mod_php71 package to be safe. It only contains the module and some license files:

```
$ pkg info -l mod_php71

mod_php71-7.1.17:


/usr/local/libexec/apache24/libphp7.so


/usr/local/share/licenses/mod_php71-7.1.17
/LICENSE


/usr/local/share/licenses/mod_php71-7.1.17
/PHP301


/usr/local/share/licenses/mod_php71-7.1.17
/catalog.mk
```

Meet the Author



Bob Cromwell has been using OpenBSD since, well, not sure how long… Some time in the late 1990s. He's used Linux since you downloaded 40+ floppy images, some time around 1993-1994. Before that he had used UNIX, SunOS and forms of BSD, at Purdue since the mid 1980s. He got a BSEE at Purdue back then, worked at the university, grad school, Ph.D. in electrical and computer engineering, has done consulting since 1992. He's taught courses for Learning Tree International since the mid 1900s, and has written courses for them since the late 1990s. He's a more recent convert to FreeBSD.

MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Device Driver Development for BSD

Character Devices

How to Add New System Calls to Your OS

Basic Notions About Network Drivers

Soon Online ... www.bsdmag.org

VOL 12 NO 05
ISSUE 05/2018 (105)
ISSN 1898-9144

Self Exposure

Redundant Firewalls with OpenBSD, CARP and pfsync

Daniele Mazzocchio
Applies to: OpenBSD 6.3
Last update: May 29, 2018

http://www.kernel-panic.it/openbsd/carp/

Table of contents

1. Introduction

Firewalls are among the most critical components in network infrastructure, since their failure may cause entire groups of machines to go offline. The damage may range from the public (web, mail, DNS, etc.) servers to become unreachable from the outside world up to being unable to surf this website!

Using firewall clusters can dramatically reduce these risks, making the failure of a firewall completely transparent to users. Also, maintenance (patching, upgrading, rebooting, etc.) becomes much easier and faster when relying on a backup machine, thus indirectly increasing systems security and reliability.

On the other hand, it's true that redundancy increases hardware costs and can't solve every problem, like transparent transfer of certain protocols (e.g., SSH or IRC) between systems or synchronizing data between clustered machines (as a matter of fact, we will rely on two different protocols for failover and synchronization).

The tools we will use to build our failover cluster are:

OpenBSD

it is largely considered one of the most secure OSes around, with *only two remote holes in the default install, in a heck of a long time!*;

Packet Filter (PF)

OpenBSD's system for filtering TCP/IP traffic and doing Network Address Translation;

CARP (Common Address Redundancy Protocol)

the protocol that achieves system redundancy, by having multiple computers creating a single, virtual network interface between them;

pfsync

the protocol that allows PF state tables to be synchronized between multiple firewalls.

A good knowledge of OpenBSD and PF is assumed, since we won't cover topics like pfctl(8) and pf.conf(5) syntax. Anyway, the appendix contains some useful links for more on these topics.

2. Network layout

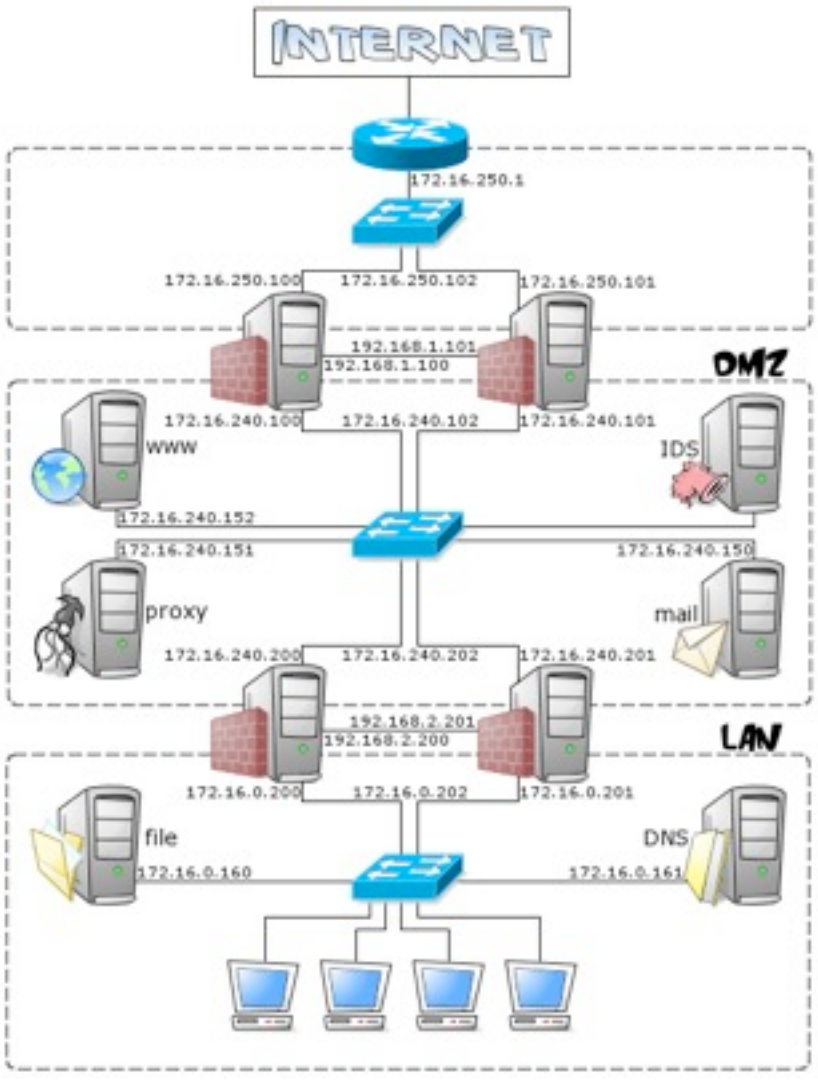
First, let's take a look at the environment in which our firewall clusters will operate. It's a very simple and "classic" network, made up of:

a DMZ (172.16.240.0/24), containing the publicly accessible machines (e.g. web and mail servers) and the intrusion detection sensors;

a LAN (172.16.0.0/24), containing clients and servers not accessible from the public Internet (file server, DHCP server, internal DNS server, etc.);

a router, in a small subnet (172.16.250.0/24), to connect the network to the Internet.

This environment requires that we set up two firewall clusters: the first separating the DMZ from the Internet (we won't take into account any router filtering); the second between the LAN and the DMZ. The network looks roughly like this:



The great advantage of this topology is that it needs two firewall clusters, thus allowing us to look over two slightly different cluster configurations. Jokes apart, these are some of its major benefits:

in case of a firewall compromise, the LAN is protected by an additional layer of filtering (though it would be better to use different firewall platforms, to prevent attackers from compromising the internal firewalls with the same technique);

a single (though clustered) firewall, filtering both LAN and DMZ traffic, is a single point of failure;

on each firewall, rules apply only to LAN or DMZ traffic, thus making PF rulesets cleaner and easier to maintain;

but there are also a few drawbacks:

besides its traffic, the DMZ must support the traffic load from the internal network to the Internet;

double-filtering LAN traffic increases security but (slightly) affects performances;

the cost of additional hardware may not be irrelevant.

3. Base configuration

Let's take a brief look at the base system configuration, which applies to all of our firewalls.

We won't go through the installation of the operating system, which is very well documented on the OpenBSD web site. The only note is that you should install only the bare minimum, to prevent firewall security and reliability from being compromised by unnecessary software. Therefore, during installation, you only need to select file sets marked as ***Required*** by the documentation, i.e.:

BSD, the kernel;

baseXX.tgz, the base system;

There should be no need to install the compiler (compXX.tgz), also to avoid providing such a useful tool to possible intruders (see [PUIS]).

After the first reboot, we can start doing some basic configuration; by default, OpenBSD doesn't start unnecessary daemons, though I guess we can stop sndiod(8) on a firewall. It's also a good practice to edit the /etc/motd file to give as few information as possible about the system and to warn users, whether legitimate or not, that all access is being logged and that any unauthorized access will be prosecuted (see [PUIS]).

You should already have configured the network during installation; anyway, if you need to make some changes, these are the main files to edit:

/etc/hostname.if(5)

containing information regarding the configuration of each network interface (address, netmask, etc.);

/etc/mygate(5)

containing the address of the gateway host;

/etc/myname(5)

containing the symbolic hostname (FQDN) of the machine;

/etc/resolv.conf(5)

containing the resolver configuration settings (name servers, local domain name, etc.).

Considering the large amount of DNS-based attacks, it is also preferable, especially on firewalls, not to rely on DNS to resolve names and addresses of the most critical systems, but rather inserting them into the /etc/hosts(5) file; to make sure this file has a higher priority than DNS, just make sure that /etc/resolv.conf(5) contains the line:

```
/etc/resolv.conf

lookup file bind
```

Packet Filter is enabled by default, and loads its rules from the /etc/pf.conf(5) file. You may also want to change the pflogd(8) flags in the variable pflogd_flags. Lastly, don't forget to enable IP and IPv6 forwarding by issuing the command:

```
# sysctl net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
# sysctl net.inet.ip6.forwarding=1
net.inet.ip6.forwarding: 0 -> 1
#
```

and to add the following lines to /etc/sysctl.conf(5) to re-enable forwarding after reboot:

```
/etc/sysctl.conf

net.inet.ip.forwarding=1
net.inet.ip6.forwarding=1
```

4. The CARP protocol

CARP (Common Address Redundancy Protocol) is the protocol that achieves system redundancy by sharing an IP address across a group of hosts on the same network segment (redundancy group). When one of these hosts becomes unavailable, another host in the redundancy group takes over, with no loss of network traffic. CARP also allows a degree of load sharing between systems.

Although creating redundant firewalls is one of its most common uses, CARP isn't a firewall-specific protocol. It can be used to ensure service continuity and/or load sharing to a number of network services.

Anecdotally, the OpenBSD team wanted to produce a free implementation of the IETF standard protocols, VRRP (Virtual Router Redundancy Protocol), defined in [RFC3768], and HSRP (Hot Standby Router Protocol), defined in [RFC2281]. However, Cisco claiming patent rights on it *firmly informed the OpenBSD community that Cisco would defend its patents for VRRP implementation* (see [CARP] for more details). This move forced the OpenBSD developers to create a new, competing protocol designed to be fundamentally different from VRRP. And the funny thing is, *putting CARP hosts on a network with Cisco VRRP hosts made Cisco routers crash* [LUCAS].

CARP is a multicast protocol, grouping several physical computers together under one or more virtual addresses. Of these, one system is the master and responds to all packets destined for the group; the other systems (backups) just stand by, waiting for any problem to take its place (as it happens between co-workers).

At configurable intervals, the master advertises its operation on IP protocol number 112. If the master goes offline, the other hosts in the redundancy group begin to advertise. The host that can advertise most frequently becomes the new master. When the main system comes back up, it becomes a backup host by default, although it can be configured to try to become master again.

As you can see, CARP only creates and manages the virtual network interface. It's up to the system administrator to synchronize data between applications, using pfsync(4) (which we'll discuss in the next chapter), rsync or whatever protocol is appropriate for the specific application.

4.1 Configuration parameters

CARP configuration is done via the sysctl(8) and ifconfig(8) commands. There are three relevant sysctl(2) variables:

net.inet.carp.allow

defines whether the host handles CARP packets or not. It is enabled by default;

net.inet.carp.log

defines whether to log CARP messages or not. It may be a value between 0 and 7, corresponding to the syslog(3) priorities, and defaults to 2 (i.e. only CARP state changes are logged);

net.inet.carp.preempt

if set to 0 (default), the host won't try to become master when it receives CARP advertisements from another master. Otherwise, it will try to become master if it can advertise more frequently than the current master. This option also enables failing over all interfaces if one interface goes down. In fact, if one physical CARP-enabled interface goes down, CARP will increase the demotion counter by 1(see below) for all groups that the interface belongs to, thus allowing the election of new masters on all subnets.

The syntax for configuring CARP with ifconfig(8) is:

```
   ifconfig carpN create

    ifconfig carpN [advbase n] [advskew n]
[balancing mode]   \
    [carpnodes
vhid:advskew,vhid:advskew,...] [carpdev
iface] \
    [[-]carppeer peer_address] [pass
passphrase] [state state] [vhid host-id]
```

carpN

the name of the carp(4) virtual interface.

advbase, advskew

these values determine the interval between two consecutive CARP advertisements. This interval (in seconds) is given by the formula (advbase + (advskew / 255)); increasing advbase will decrease network traffic, but increase the delay in electing the new master. Small advskew values allow a host to advertise more frequently, increasing its probability to become master. The values of advbase and advskew must be in the range of 0 to 254 and default to 1 and 0 respectively;

balancing

sets the load balancing mode (which will be discussed later); valid modes are ip, ip-stealth and ip-unicast;

carpnodes

a comma-separated list of vhid:advskew pairs that actually define how the load should be shared among the configured carp nodes (see below for further details);

carpdev

specifies the physical interface that belongs to this redundancy group. By default, CARP uses the physical interface on the same subnet as the virtual interface;

[-]carppeer

allows you to specify the IP address of the other CARP peer(s), instead of using the default multicast group; this allows the use of ipsec(4) to protect carp(4) traffic;

pass

the authentication password to use when talking to other CARP-enabled hosts in the redundancy group. This must be the same on all members of the group;

state

force a carp(4) interface into a specific state (init, backup or master);

vhid

the Virtual Host ID. This is a unique number (between 1 and 255) that is used to identify the redundancy group to the other nodes on the network.

4.1.1 The demotion counter

Besides basic configuration, the ifconfig(8) command also allows you to tweak the CARP demotion counter, which is *a measure of how "ready" a host is to become master of a CARP group* [CARPFAQ] (the higher the counter, the less ready the host). Let's see it in more detail.

CARP interfaces are divided into groups (by default all carp(4) interfaces are members of the "carp" interface group), and each group is assigned a demotion counter, whose value can be viewed by running the following command:

$ **ifconfig -g carp**
carp: carp demote count 0

The demotion counter comes in handy mainly when:

You want to momentarily prevent a host from becoming master. For instance, at boot time, the rc(8) script increases the demotion counter by 128 before starting the network, and decreases it by the same amount once all interfaces have been initialized and all network daemons have been started (the demotion counter can't be set to an absolute value, but only increased or decreased by a certain amount):

```
/etc/rc

ifconfig -g carp carpdemote 128
[ ... ]
ifconfig -g carp -carpdemote 128
```

You want to gracefully failover only a limited number of a host's carp(4) interfaces (not all of them, as it happens when an interface goes down and preempt is enabled). In the following example, we will failover the carp1 and carp2 interfaces and leave the state of the others unchanged:

```
# ifconfig carp1 group morituri
# ifconfig carp2 group morituri
# ifconfig morituri
carp1:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MU
LTICAST> mtu 1500
        carp: MASTER carpdev sis0 vhid 1
advbase 1 advskew 100
        groups: carp morituri
        inet 1.2.3.4 netmask 0xffffff00
broadcast 1.2.3.255
carp2:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MU
LTICAST> mtu 1500
        carp: MASTER carpdev sis1 vhid 2
advbase 1 advskew 100
        groups: carp morituri
        inet 2.3.4.5 netmask 0xffffff00
broadcast 2.3.4.255
# ifconfig -g morituri
morituri: carp demote count 0
# ifconfig -g morituri carpdemote 50
```

```
# ifconfig -g morituri
morituri: carp demote count 50
```

For further details on the CARP demotion counter, please refer to [CARPFAQ].

4.1.2 Load balancing

CARP allows you to load balance incoming network traffic among a set of CARP-enabled hosts. First, you need to create a load balancing group by configuring, on each balanced carp(4)interface, as many VHIDs as hosts in the balancing group. The advskew on each VHID will be configured so that each host will be the master on a separate VHID (see below for a practical example).

Since balancing requires that all CARP hosts receive network traffic destined to the CARP address, the virtual interface will use a multicast MAC address, forcing the switch to send incoming traffic to all nodes in the redundancy group.

CARP uses the hash of the source and destination addresses of the IP packet to determine which VHID (and therefore which host) should accept the packet; balancing can be enabled using ifconfig(8), by setting the balancing option to "ip". For example:

```
# ifconfig carp0 balancing ip carpnodes
1:0,2:100
```

Alternatively, you can set the balancing option to "ip-stealth" (stealth mode), to prevent hosts from sending packets with their virtual MAC address as source; this will prevent the switch from learning the virtual MAC address, forcing it to flood the traffic to all its ports. Last, if you're using a hub or a switch that supports some kind of monitoring mode, you can set balancing to "ip-unicast".

4.2 Parameters configuration

Now it's time to configure CARP on our firewalls. To examine two slightly different CARP configurations, we will set up the two internal firewalls (Mickey and Minnie, between LAN and DMZ) in active/standby mode, with only one system filtering the whole network traffic, and the other one acting as a hot spare. The two external firewalls (Donald and Daisy, separating the DMZ from the internet), instead, will be in active/standby mode, sharing the traffic load.

So let's recap the firewalls addresses, as we have seen them in the network diagram:

	Mickey	Minnie	Virtual address
LAN	172.16.0.200	172.16.0.201	172.16.0.202
DMZ	172.16.240.200	172.16.240.201	172.16.240.202
pfsync	192.168.2.200	192.168.2.201	
	Donald	**Daisy**	**Virtual address**
DMZ	172.16.240.100	172.16.240.101	172.16.240.102
Internet	72.16.250.100	172.16.250.101	172.16.250.102
pfsync	192.168.1.100	192.168.1.101	

4.2.1 Active/standby configuration

Let's start with Mickey and Minnie: first, we need to create the carp* devices and configure them with ifconfig(8):

```
mickey# ifconfig carp0 172.16.0.202/24
vhid 1 pass password1 advbase 1 advskew 0
mickey# ifconfig carp1 172.16.240.202/24
vhid 2 pass password2 advbase 1 advskew 0
```

```
minnie# ifconfig carp0 172.16.0.202/24
vhid 1 pass password1 advbase 1 advskew
100
minnie# ifconfig carp1 172.16.240.202/24
vhid 2 pass password2 advbase 1 advskew
100
```

We have just created the interfaces, assigned them an IP address, a virtual host ID (1 on the

LAN, 2 on the DMZ), and a password (probably not the most secure) for authentication. We also decided that, whenever possible, Mickey will be the master; this is done by giving Minnie a higher advskew value (100), thus making the interval between its advertisements (1 + 100 / 255) higher than the interval between Mickey's advertisements (1 + 0 / 255). Additionally, as we've seen above, the host that's able to advertise most frequently becomes master.

Furthermore, by setting net.inet.carp.preempt to "1" on Mickey, we ensure that Mickey will always try to become the master:

```
mickey# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1
```

To make these settings permanent after reboot, we need to edit the /etc/hostname.carp* and /etc/sysctl.conf files on Mickey:

```
/etc/hostname.carp0

inet 172.16.0.202 255.255.255.0
172.16.0.255 vhid 1 pass password1 advbase
1 advskew 0


/etc/hostname.carp1

inet 172.16.240.202 255.255.255.0
172.16.240.255 vhid 2 pass password2
advbase 1 advskew 0


/etc/sysctl.conf

[...]
net.inet.carp.preempt=1


and on Minnie:

/etc/hostname.carp0

inet 172.16.0.202 255.255.255.0
172.16.0.255 vhid 1 pass password1 advbase
1 advskew 100


/etc/hostname.carp1

inet 172.16.240.202 255.255.255.0
172.16.240.255 vhid 2 pass password2
advbase 1 advskew 100
```

Note: to make the adoption of CARP easier on pre-existing networks, CARP allows using the physical address of a host as the virtual address of the whole redundancy group.

4.2.2 Active/active configuration

Now let's get on to Donald and Daisy, and start by configuring their DMZ interfaces. As before, we will create the carp0 device on each machine, but this time, to enable load balancing, we will use the carpnodes option to assign two different Virtual Host IDs to the interface (VHIDs 3 and 4).

On VHID 3, we will set the advskew of Donald and Daisy to 0 and 100 respectively: this will ensure that Donald becomes master for that VHID; on VHID 4, we will do the opposite, by setting the advskew of Donald and Daisy to 100 and 0 respectively to force Daisy to become master for VHID 4:

```
donald# ifconfig carp0 172.16.240.102/24
balancing ip carpnodes 3:0,4:100 \
> pass password3
donald# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1


daisy# ifconfig carp0 172.16.240.102/24
balancing ip carpnodes 3:100,4:0 \
> pass password3
daisy# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1
```

We now have two redundancy groups with the same IP address, but each with a different master:

```
donald# ifconfig carp0
carp0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MU
LTICAST> mtu 1500
```

```
        lladdr 01:00:5e:00:01:01
        carp: carpdev rl1 advbase 1
balancing ip
                state MASTER vhid 3
advskew 0
                state BACKUP vhid 4
advskew 100
        groups: carp
        inet 172.16.240.102 netmask
0xffffff00 broadcast 172.16.240.255
        inet6
fe80::2c0:a8ff:fe8e:b112%carp0 prefixlen
64 scopeid 0x5
```

```
daisy# ifconfig carp0
carp0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MU
LTICAST> mtu 1500
        lladdr 01:00:5e:00:01:01
        carp: carpdev rl1 advbase 1
balancing ip
                state BACKUP vhid 3
advskew 100
                state MASTER vhid 4
advskew 0
        groups: carp
        inet 172.16.240.102 netmask
0xffffff00 broadcast 172.16.240.255
        inet6
fe80::219:d2ff:fe02:6469%carp0 prefixlen
64 scopeid 0x5
```

To make these settings permanent across reboots, we need to edit the startup files on Donald:

```
/etc/hostname.carp0

inet 172.16.240.102 255.255.255.0
172.16.240.255 balancing ip carpnodes
3:0,4:100 pass password3


/etc/sysctl.conf

[...]
net.inet.carp.preempt=1
```

and Daisy:

```
/etc/hostname.carp0

inet 172.16.240.102 255.255.255.0
172.16.240.255 balancing ip carpnodes
3:100,4:0 pass password3


/etc/sysctl.conf

[...]
net.inet.carp.preempt=1
```

Now we have to do the same on the external network interfaces, with another two Virtual Host IDs (VHIDs 5 and 6):

```
donald# ifconfig carp1 172.16.250.102/24
balancing ip carpnodes 5:0,6:100 \
> pass password5
```

```
daisy# ifconfig carp1 172.16.250.102/24
balancing ip carpnodes 5:100,6:0 \
> pass password5
```

and edit the startup files on Donald:

```
/etc/hostname.carp1

inet 172.16.250.102 255.255.255.0
172.16.250.255 balancing ip carpnodes
5:0,6:100 pass password5
```

and Daisy:

```
/etc/hostname.carp1

inet 172.16.250.102 255.255.255.0
172.16.250.255 balancing ip carpnodes
5:100,6:0 pass password5
```

Though the above configuration involves only a couple of machines, it can be easily extended to up to 32 hosts. *One last note*: load sharing won't probably achieve a perfect 50/50 distribution between the two machines, since CARP uses a hash of the source and destination IP addresses to determine which system should accept a packet, not the actual load.

5. The pfsync protocol

Pfsync is the protocol used by Packet Filter to manage and update state tables, which allow for stateful inspection and NAT. By default, state change messages are sent out on the synchronization interface using IP multicast packets. The protocol is IP protocol 240 and the multicast group used is 224.0.0.240. We will use it to synchronize state tables among firewalls of the same redundancy group and, in the event of a failover, allow network traffic to flow uninterrupted through the new master firewall.

pfsync(4) is also the name of the pseudo-device on which PF state table changes are sent (except states created by rules marked with the no-sync keyword or by pfsync(4) packets). pfsync(4) can be configured to use a physical interface in order to merge and keep in sync the state tables among multiple firewalls.

The physical synchronization interface can be set through ifconfig(8), using the syncdev parameter. For example, on our firewalls, we can write:

```
# ifconfig pfsync0 syncdev rl2
```

assuming that the rl2 interface is, on each host (see picture), the interface on the 192.168.1.0/24 subnet (for Mickey and Minnie) or 192.168.2.0/24 (for Donald and Daisy), and cross-cabled to the "beloved" firewall.

Crossover cables are recommended because the pfsync protocol doesn't provide any cryptography or authentication mechanism. If you don't use a secure network, like a crossover cable, an attacker may use spoofed pfsync packets to alter the firewalls state tables and bypass filter rules.

Alternatively, you can use the syncpeer keyword to specify the address of the firewall to synchronize with. The system will use this address, instead of multicast, as the destination of pfsync(4) messages, allowing the use of IPsec to protect the communication. In this case, syncdev must be set to the enc(4) pseudo-device, which encapsulates/decapsulates ipsec(4) traffic. E.g.:

```
# ifconfig pfsync0 syncpeer 192.168.1.101
syncdev enc0
```

To make these settings permanent after reboot, we need to edit the /etc/hostname.pfsync0 file on each firewall:

```
/etc/hostname.pfsync0

up syncdev rl2
```

6. PF rules!

The impact of CARP and pfsync on Packet Filter rules is minimal. First, you need to let the PFSYNC and CARP protocols pass on their interfaces:

```
pass quick on rl2 proto pfsync keep state
(no-sync)
pass on { rl0, rl1 } proto carp keep state
(no-sync)
```

Then, when writing firewall rules, keep in mind that, from pf(4)'s point of view, all traffic passes through the physical interface. Therefore, in cases like:

```
pass in on $ext_if [...]
```

you can keep referring to the physical, not the virtual interface.

On the other hand, the virtual address is associated to the CARP interface; thus, you need to refer to it if the firewall offers any services on its virtual address:

```
# SSH on the virtual interface
pass in on $int_if inet proto tcp from
$int_if:network to carp0 port ssh
```

or on a NATed server, through traffic redirection:

```
# Mail server accessible from the internet
pass in on $ext_if inet proto tcp from any
to carp2 port $mail_ports rdr-to $mail_srv
```

In all other cases, CARP is perfectly transparent to pf(4), as for services offered by the firewall on its physical addresses:

```
# SSH on the physical address
pass in on $int_if inet proto tcp from
$int_if:network to $int_if port ssh
```

or for normal filtering:

```
# External DNS
pass in  on $int_if inet proto { tcp, udp
} from $int_if:network to $dns_srv \
     port domain
pass out on $ext_if inet proto { tcp, udp
} from $ext_if to $dns_srv \
     port domain
```

As an example, let's see a basic PF ruleset for our external firewalls, Donald and Daisy:

```
##############################################
# Macros and lists
#
##############################################
##################################

ext_if  = rl0                           #
External interface
int_if  = rl1                           # DMZ
interface
pfs_if  = rl2                           #
Pfsync interface
carp_if = carp1                         #
External CARP interface

mail_srv = "mail.kernel-panic.it"
# Mail server
web_srv  = "{ www1.kernel-panic.it,
www2.kernel-panic.it }"     # Web servers
dns_srv  = "{ dns1.isp.com, dns2.isp.com }"
# DNS servers
int_fw   = "{ mickey.kernel-panic.it,
minnie.kernel-panic.it }" # Internal fw

mail_ports = "{ smtp, submission, imap, imaps
}"    # Mail server ports
web_ports  = "{ www, https }"
# Web server ports

# Allowed incoming ICMP types
icmp_types = "{ echoreq, timex, paramprob,
unreach code needfrag }"

# Private networks (RFC 1918)
priv_nets = "{ 127.0.0.0/8, 10.0.0.0/8,
172.16.0.0/12, 192.168.0.0/16 }"

##############################################
##################################
# Options, scrub and NAT
#
##############################################
##################################

set block-policy drop
set loginterface $ext_if
set syncookies always
set skip on lo

# NAT outgoing connections
match out on $ext_if from !$ext_if to any
nat-to $ext_if

# Redirect web services (with load balancing)
match in on $ext_if inet proto tcp from any to
$carp_if port $web_ports \
    rdr-to $web_srv round-robin sticky-address

# Redirect mail services
match in on $ext_if inet proto tcp from any to
$carp_if port $mail_ports \
    rdr-to $mail_srv

##############################################
##################################
# Filtering rules
#
##############################################
##################################

block all                               # Default
deny
block in quick from urpf-failed     # Spoofed
address protection

# Scrub incoming packets
```

```
match in all scrub (no-df)

pass quick on $pfs_if proto pfsync keep state
(no-sync)         # Enable pfsync
pass on { $int_if, $ext_if } proto carp keep
state (no-sync)    # Enable CARP

block in  quick on $ext_if from $priv_nets to
any
block out quick on $ext_if from any to
$priv_nets

# Mail server
pass in  on $ext_if inet proto tcp from any to
$mail_srv port $mail_ports
pass out on $int_if inet proto tcp from any to
$mail_srv port $mail_ports
pass in  on $int_if inet proto tcp from
$mail_srv to any port smtp
pass out on $ext_if inet proto tcp from
$ext_if to any port smtp modulate state

# Web servers
pass in  on $ext_if inet proto tcp from any to
$web_srv port $web_ports \
     synproxy state
pass out on $int_if inet proto tcp from any to
$web_srv port $web_ports

# ICMP
pass in  inet proto icmp all icmp-type
$icmp_types
pass out inet proto icmp all

# DNS
pass in  on $int_if inet proto { tcp, udp }
from $int_if:network to $dns_srv \
     port domain
pass out on $ext_if inet proto { tcp, udp }
from $ext_if to $dns_srv \
     port domain

# Internet web servers
pass in  on $int_if inet proto tcp from
$int_fw to any port $web_ports
pass out on $ext_if inet proto tcp from
$ext_if to any port $web_ports \
     modulate state
```

7. Appendix

7.1 References

[PUIS] - *Practical UNIX and Internet Security*, Simson Garfinkel and Gene Spafford, O'Reilly, 2003

[CARP] - "CARP License" and "Redundancy must be free"

[LUCAS] - Cisco supports CARP? Ha ha ha hahaha?

[PFFAQ] - PF: The OpenBSD Packet Filter

[RFC3768] - RFC 3768, Virtual Router Redundancy Protocol (VRRP)

[RFC2281] - RFC 2281, Cisco Hot Standby Router Protocol (HSRP)

[CARPFAQ] - PF: Firewall Redundancy with CARP and pfsync

7.2 Bibliography

[ABSO] - *Absolute OpenBSD*, Michael W. Lucas, No Starch Press, 2003

The Common Address redundancy Protocol (CARP)

Firewall Failover with pfsync and CARP

Firewalling with OpenBSD's PF packet filter, Peter N. M. Hansteen, 2008

The Book of PF, 2nd Edition, Peter N. M. Hansteen, No Starch Press, 2010

The OpenBSD PF Packet Filter Book, Jeremy C. Reed, Reed Media Services, 2006

Meet Daniele Mazzocchio

Please tell us about yourself?

I'm a Unix/Linux system administrator, Python developer, and a long-time OpenBSD fan.

I started a website, www.kernel-panic.it , a few years ago, where I get to share some of my OpenBSD documentation and software projects.

I've also written a Python module, py-pf, that provides bindings for OpenBSD's Packet Filter.

Please, let us know more about your website and blog. What is the most interesting topic which can be found?

I guess the most interesting topic is OpenBSD system administration. I started writing about OpenBSD mainly for two reasons: to keep track of my experiments with network services on OpenBSD and to persuade other people, including friends and colleagues, that OpenBSD can do everything Linux does, and better!

I think that making a website that is at least useful to me, made it useful to other people as well and I hope it made someone's life easier.

In every article, before diving into the configuration details, I also try to give some technical background on the software and protocols I use (such as IPsec, DNS, and LDAP). I think this information helps readers to tweak the proposed configuration to fit their needs and requirements better.

Reading your blog, we can see that you have a wide field of expertise. Tell us more about your experience. Also, is OpenBSD your favorite OS?

When I first fell in love with computers, all I wanted to do was to learn as much as I could about programming and operating systems. So I started reading lots of documentation, RFCs, books, and installed my first Linux system.

Linux looked like the perfect playground to learn and experiment. It had so many powerful tools, software, programming languages. I was also very fascinated by the Open-Source community for its strong ethics, sharing of knowledge, and love of freedom.

And one day I discovered OpenBSD, the "secure by default" OS, and fell in love with it and its philosophy.

Why? What are your best features, the ones you like the most?

I've always been very concerned about security. Therefore, when I first heard about OpenBSD and its security-centric approach, I definitely wanted to give it a try; and I was pretty amazed at how OpenBSD developers managed to combine security with usability and simplicity.

My favourite feature is Packet Filter, combined with CARP and pfsync for redundancy. It's the best firewall I've ever worked with.

I'm also a very happy user of httpd(8) and smtpd(8), which provide lightweight replacements for nginx and sendmail. Additionally,I love the OpenBSD installer because it’s minimal, fast, and simple.

What is the most interesting programming issue you’ve encountered?Why was it so amazing?

Finding one is hard. All issues look trivial once you find the solution!

Anyway, one of my first experiments with Python has been writing my email client: this was long ago, so I can’t remember all the issues I encountered. However, I recall how each of them

forced me to read more documentation, RFCs, source code; and all that research made me learn a lot about programming, network protocols, and standards.

Is Python your preferred language? Why would you choose it over other programming languages?

I have experimented with a few programming languages, but I'm still most comfortable with Python.

It is very readable, elegant and has a library for everything; and unlike many of its critics, I never experienced performance or memory footprint issues with Python.

What was the most challenging implementation you've done so far?

Shieldcore, a Canadian company, asked me to collaborate on the creation of an OpenBSD-based firewall appliance a few years ago.

I was in charge of the backend development, i.e., the Python layer that allowed communication between the web frontend and the Operating System.

The challenge was to develop a full set of Python bindings for OpenBSD and provide a complete API to the web frontend. It's been a lot of work, but it also gave me the opportunity to read lots of OpenBSD code and to learn a lot from it!

What are your favourite features in the new releases of OpenBSD?

I'm very grateful to Antoine Jacoutot for writing syspatch(8), which makes it so much easier to apply security errata. I'm also very excited about the development of vmd(8), which I extensively use in my test environments.

I also love features like pledge(2) and KARL (Kernel Address Randomized Link), that show how much OpenBSD developers' do to improve system security.

Do you have any specific goals for the rest of this year?

I'd like to update and release parts of the code I developed with Shieldcore; this would be the first building block of an interface for making management and monitoring of my OpenBSD systems easier and faster.

I'd also like to learn to play the violin.

What's the best advice you can give to the BSD magazine readers?

Always keep a rubber duck next to your computer (personally, I have a pufferfish); whenever you have an issue that seems impossible to solve, talk to the duck about it. The solution will come sooner than you'd expect. It always works for me.

Thank you

Thanks for reading.

Expert Speak by E.G. Nadhan

Just Takes 5 Seconds to Grow Your Team Culture

How many times have you been in a situation where you are about to sharply critique a co-worker, a colleague, an acquaintance for something they did not do right? Well, as it turns out, Gallup's workplace research suggests praise should outweigh criticism by a 5-to-1 margin. Five praises for one criticism (if at all there is one). Chester Elton points this out in his article The 5-to-1 ratio that can change your team culture. This is easier said than done. How can we ensure that we exercise some control over the urge to critique? Well, say hello to the 5-second rule from Mel Robbins. Join me as I walk you through how this rule could be applied to balance praise versus criticism to grow a positive team culture.

Lately, I came across two powerful influencers in a span of a few weeks. Both influencers are strong advocates of the principles they embrace. I am just fascinated by how one influencer's principle could be applied to the other with the goal of evolving a team culture that is most conducive to a healthy and productive environment.

Principle 1: *5-second rule.* Mel Robbins, a serial entrepreneur and one of the most booked motivational speakers in the world, advocated the **5-second** rule. In 2017, Mel broke self-publishing records with her international best-seller, The 5 Second Rule. It was named the #1 audiobook in the world and the fifth most read book of the year on Amazon. It is translated into 31 languages. The principle itself is very simple. At those critical moments when you know what the right thing to do is but you find one excuse or the other to rationalize and avoid doing it, take a deep breath and count down **5 - 4 - 3 - 2 - 1** and magic happens! **Key takeaway:** Take pause!

Principle 2: *5-to-1 ratio.* People want to know that their bosses see their effort and truly value it, says Elton. This ties to feelings of job security and well-being. Employees also need the affirmation that their co-workers see them as trustworthy, dependable, and creative. This reinforces that you have friends at work, acceptance, and that others have your back. This is an ecosystem of psychological safety that mitigates natural infighting and jealousies. An ecosystem that is reinforced with the 5-to-1 ration for praise versus criticism. **Key takeaway:** Self-control.

Adrian Gostick and Chester Elton provide real solutions for managing culture change, driving innovation, and leading a multi-generational workforce. Known as The Carrot Guys, they are authors of the #1 New York Times, USA Today and Wall Street Journal bestsellers All In and The Carrot Principle. Their books have been translated into 30 languages and have sold 1.5 million copies around the world. So, how do these Principles come together? How can we apply them to be more effective in our interactions in the workplace and even in our personal lives?

Here is how I plan on doing it.

Next time I make an observation that is likely to come across as a criticism of a fellow worker, a colleague, a leader or a mentee, I am going to think of Mel Robbins and count down **5 - 4 - 3 - 2 - 1** to see if this criticism is really called for. Are there other ways that the same message can be delivered in a more positive tone? Once I count down and apply the 5-second rule, I am less likely to share the criticism. Similarly, when I want to praise someone, I am **not going to apply** the 5-second rule. Praise has a much more open avenue than criticism.

In an open culture, constructive criticism delivered with the overall team and project goals in mind is a good practice. Even so, the 5-second rule can help us maintain a balance on ourselves. The rule also gives us an opportunity to give a second, a third, a fourth and a fifth ... thought to an otherwise spontaneous act. There you go. Two influencers. Two powerful principles, which can be applied in tandem the spirit of a better team culture.

What's your say? What is your ratio for praise versus criticism? Have you tried the 5-second rule? If not, now is the time!

Here is what I say: **5 - 4 - 3 - 2 - 1 -- Zero -- Blast off to Team Culture!**

Meet the Author

E.G. Nadhan is the Chief Technology Strategist for the Central Region at Red Hat. He provides thought leadership on various concepts including Cloud, Big Data, Analytics and the Internet of Things (IoT) through multiple channels including industry conferences, Executive Roundtables as well as customer specific Executive Briefing sessions. With 25+ years of experience in the IT industry selling, delivering and managing enterprise solutions for global corporations, he works with the executive leadership of enterprises to innovatively drive Digital Transformation with a healthy blend of emerging solutions and a DevOps mindset. Follow Nadhan on Twitter and LinkedIn.

Interview with

Joel Knight

Please tell us about yourself?

I live and work in Calgary, Canada (which is also home to the OpenBSD project). I'm a network engineer by profession but have been using and involved in open source software as a hobby for many years now. I never really got into gaming. I'm not very artsy. I like to build things. Building and maintaining systems is my happy place. Luckily I don't have to do that for work, so it's strictly something I do for myself and fun. I'm an original contributing author to the OpenBSD PF User's Guide (http://www.openbsd.org/faq/pf) and the original author of some of the native OpenBSD SNMP MIBs (packetmischief.ca/openbsd-snmp-mibs, cvsweb.openbsd.org/cgi-bin/cvsweb/src/share/snmp/). I've contributed some minor patches to the OpenBSD pf(4) subsystem and network stack over the years. I have a technology blog at www.packetmischief.ca where I write about systems, software, code, security, and networks. I'm also on Twitter as @knight_joel where I tweet about the same. My Github is github.com/knightjoel.

How you first got involved with programming? What was your path?

I should state outright that I don't consider myself a programmer :-). I watch the commits that go into OpenBSD and FreeBSD: those folks are programmers. I'm a hobbyist who enjoys fixing problems and creating useful code (usually for myself and then shared as open-source). I was fortunate enough to be exposed to introductory C while still in high school. I loved being able to build software with my own hands (my favorite toy as a kid was LEGO), and so I took some additional courses but it was nothing like what kids have access to in school now. The courses were elementary, and the instructors often knew only slightly more than the students, if we were lucky. However, everything changed when I got my first Linux CD and started delving into the open-source world. "Wow, there's a full C/C++ compiler on here!" "Wow, there are other languages like Perl on here!" "Wow, I can learn a lot more now!" So from there, I got some books and started using Perl which wasn't a steep learning curve because I was able to reuse a lot of the constructs I learned with C (conditionals, control flow, etc.). And from there it was just natural to learn a bit of shell, PHP when it came along, and most recently, Python. The thrill of building things has motived me the whole time.

Reading your profile we can see that you have a wide field of expertise. Please tell us which area is your favourite one?

Yes, I'm interested in a lot of areas. I like pretty much anything to do with IT infrastructure. Or put another way, anything that isn't user-facing :-) Network engineering is the area I have the most

experience and training, and that's what's put food on my table for many years now. I like networking a lot. Recently, I've shifted my career to the cloud where I'll have to hone my skills in new areas such as databases, analytics, and AI. I can't say I have a specific favorite. I like learning new skills, and I'm not content saying "oh this one is my favorite so I'm just going to concentrate on this now".

What is your favourite OS? Why? What features are the best and what you like the most?

Windows 10, obviously ;-)

Again I'm going to kind of sit on the fence and say I don't have an absolute favorite. My general approach to things is "use the right tool for the job". I've adopted this strategy for my systems in the last couple of years: OpenBSD for my regular shell machine and any machines that need to be exceptionally hardened and have focused use-cases (i.e., DNS server, SMTP server, etc.). For machines that need to run a web stack and for the few machines I run in the cloud or as a VPS, it's FreeBSD. I just can't beat how easy it is to upgrade the OS with the freebsd-update(8) tool. And in a cloud environment where I have sketchy console access to the VM, being able to do that upgrade over an SSH session is a life saver. Sometimes I pick FreeBSD as well because it takes less time to upgrade it, and I don't want to add another OpenBSD machine to the mix which will increase my time spent doing OS maintenance. When I planned this strategy, I'm pretty sure OpenBSD didn't have the syspatch(8) utility so even just patching a remote VM was easier with FreeBSD. That's gotten really easy and stress-free with syspatch now though. I appreciate and respect the cleanliness and simplicity in OpenBSD. There's nothing extra; nothing wasted. It's focused and elegant, and it's not just because of what goes into the code, it's because of their commitment to their goals (https://www.openbsd.org/goals.html). Release after release, you always know what you're getting with OpenBSD, and that has kept me using and interested in their software. My main workstation at home is Windows 7, and I have a MacBook Pro at $WORK.

You know a lot of programming languages. What is the most interesting programming issue you encountered and why it was so amazing?

Well, I'm not sure if this is all that amazing, but it was challenging. As my first foray into Python, I was writing a web app that had a feature that allowed users to upload pictures. The app received the binary blob, stored some metadata about the picture, and then stored the image on the file system. A user could then use the web app to view the photos that had been uploaded. So I wrote the code to do this, tested it, and was pretty pleased when I got it to work. What I hadn't thought carefully about though was that the users of this app would all be on mobile devices and depending on how the user was holding the device, the image could be rotated so it appeared sideways or upside down (of course pictures always display with the correct orientation on the device because the device accounts for the rotation). This was disheartening because I felt like approximately 0% of the user base would follow instructions to only orient their phone in a specific way to work around this.

I knew I needed a way to programmatically ascertain the orientation of the image and then needed a way to rotate the image before saving it to the file system. I also knew that some image formats stored metadata about the image using Exif (https://en.wikipedia.org/wiki/Exif). Therefore, I started there and found that sure enough, rotation was a value that Exif captured. I eventually found this page (https://www.impulseadventure.com/photo/exif-orientation.html) that nicely explains the different values in the Exif rotation info. Next, though, I needed some code to parse the Exif info and manipulate the

image file. In keeping with the Don't Repeat Yourself principle, I went looking for a Python module. It doesn't take much googling to discover Pillow (http://python-pillow.org/) is the defacto Python library for working with images, and the Piexif library (http://piexif.readthedocs.io/en/latest/) is perfect for reading Exif data. In the end, the (simplified and abbreviated) code looks like this:

```
from PIL import Image
import piexif
img = Image.open(file)
if "exif" in img.info:
    exif_dict = piexif.load(img.info['exif'])
     if ('0th' in exif_dict
          and piexif.ImageIFD.Orientation in exif_dict['0th']):
        orientation = (exif_dict['0th'].pop(piexif.ImageIFD.Orientation))
        deg = None
        if orientation == 3:
            deg = 180
        elif orientation == 6:
            deg = -90
        elif orientation == 8:
            deg = 90
    if deg:
            img = img.rotate(deg, expand=True)
```

A small side note: for anyone wondering why I didn't handle image rotation in the web layer using some CSS: it's a nightmare. Some browsers will honor the rotation info in Exif out of the gate; others not. One browser honored Exif data only if you opened the raw image file in the browser and not if the image was loaded via <img> tag. Support in mobile browsers was (still is?) far from complete.

What tools do you use most often and why?

I'm very uptight about having the right tools in place on my systems. Vim is my editor of choice. I no longer remember why I chose Vim, but I keep using it because it's what I know now. I suppose most of the CLI tools I use are standard and/or boring: ssh, git, zsh, tmux. I'll mention a few I use on my workstations/iPad:

- Evernote: I use it for all kinds of things, but with respect to systems, I document each OS upgrade I do (learning from my mistakes) as well as any chronic or difficult issues I have with software, and the path I took to solve the issue. History seems to repeat itself so having a record of what I did in the past has often saved me a lot of time in the present.

- RememberTheMilk.com: Reminders and task management. I use it for all areas of my life, but again with respect to systems, I'll make a note like "upgrade such-and-such software to version X" when I see a security alert for software I use. Or if I'm doing a maintenance task on a system and I realize I should automate it, I'll make a note about how painful the task is and jot some quick ideas on how it

could be automated. This way, I'm not wasting brain cycles on keeping these tasks fresh in my memory. You could say I'm offloading them to near-line storage where I retrieve them when I'm ready to sit down and work on something.

- OpenVPN: My headend box is running OpenBSD (going back to what I said above about hardened systems), and I have Windows, FreeBSD, OS X, and Apple iOS devices as clients. I have a small script that runs after a client successfully authenticates. The script loads some pre-defined rules into a pf(4) anchor on the headend that permits/denies the client's IP address to be able to connect to certain things. With this setup, I can customize the rules that are applied to each VPN user or device.

- Zabbix: Systems always seem to break when I have the least interest in fixing them (ps, I'm never interested in fixing them). Therefore, I always monitor key metrics on everything to get as much early warning as I can that something is amiss. I'm a huge fan of Zabbix now after using Cacti for many years (https://www.packetmischief.ca/2017/02/15/why-i-enthusiastically-switched-from-cacti-to-zabbix-for-system-monitoring/)

Can you tell us about your favourite features in the new releases of your favourite OS?

Well, I'm a network nerd; hence, anything that happens around the network stack in OpenBSD usually gets my attention. For a while now, some of the network hackers have been digging deep into the network stack to move its parts out from the KERNEL_LOCK. I don't pretend to know enough of the architecture of the BSD kernel to understand what this means in detail. However, I do know that it will allow those parts of the stack to work in parallel on multiprocessor systems. This is pretty exciting for anyone that uses OpenBSD as a router or firewall, and I imagine on servers or workstations that need high network IO. I mentioned this earlier, but I'll give another shout out to the syspatch(8) utility which landed in OpenBSD 6.1. For anyone that manages multiple OpenBSD systems, the ability to pull down binary patches for the base system—that are of course cryptographically verified—is not only a huge boost to productivity but also makes managing OpenBSD systems a lot less labor intensive. My thanks to everyone who contributes to making and hosting the patches, and working on the tools to distribute and install them.

Do you have any specific goals for the rest of this year?

I took a new job with a large cloud provider this year, so most of my professional goals revolve around getting trained and learning how to properly architect solutions on the platform. Beyond that, I've got a bit of code I want to hack and always lots of blog posts to write.

What's the best advice you can give to the BSD magazine readers?

Open-source isn't an exclusive club. It's not only for advanced/ experienced programmers. It's not just for people that fit a certain demographic or stereotype. If you have the desire and the passion for being more than a user of a piece of open-source software, then that's enough. Find an area within the project where you can contribute. If I use OpenBSD as an example, people contribute to the C code (of course), but also shell code, Perl code, man page content, web site content, FAQ content, upgrade guide content, and more. Try to marry up your skills and experience with an area that the project could use and jump in! And have fun while doing it!

Thank you

Column

Online shopping and electronic transactions are revolutionizing the way business is being carried out, both for individuals and corporate entities. Are we entering a golden age of choice, or should the Latin phrase Caveat Emptor be embedded on every "accept" button for Internet sales?

by ***Rob Somerville***

I've just been ripped off for £153.25 for a Samsung Galaxy J5 mobile phone. Or to be more accurate, Amazon have, along with approximately 1,000 other customers who have paid exorbitant amounts of money to a clearly fraudulent storefront that has exploited a subtle flaw in the E-commerce model that Amazon, eBay, and PayPal operate. At the time of writing, I estimate Amazon is facing losses of £150-250K on this one storefront alone, but the total combined losses across the aforementioned platforms is well into the millions. Speaking to the Royal Mail customer services department concerning this, it is clear that carriers are under pressure due to this scam, and while I cannot remember the exact words the very pleasant and apologetic gentleman used, the impression I got is that this practice is endemic.

The scam is simple, but like all fraudulent activity, it exploits the foundations on which any civilised society is built upon – trust. First of all, you build a E-storefront that has good reviews, be it through selling to a bunch of friends or conspirators. The importance here is to get good reviews and build a good reputation. The next step is simple, offer some goods for sale slightly below market value, and wait for the orders to come in. You don't want to be too greedy, just pitch your sale slightly below the retail price, enough to attract buyers attention into thinking they have good value, but not a tremendous bargain. What you are looking for is sales quantity, but at the same time not to trigger any suspicion as to the goods being "too cheap" or possibly stolen. This is the real hook to get the customer on the line, rather than what comes next.

Once the orders start coming in, the process is simple. As payments are released to your vendor account via the trust network of being signed on delivery, dispatch the first order with the corresponding delivery ID to your accomplice, not to the person who ordered it. The carrier will flag this package as being delivered and signed for, authorize the aggregator (in this case Amazon) to release payment, and you have a credit in your account. When the original customer complains his goods were not delivered, apologize, and send a useless piece of junk (in my case, a £1 silicon telephone stand) using the delivery ID assigned to your next customer. In my case, despite being a "signed for" Royal Mail package, the carrier posted the thin padded envelope through the door, and it was not signed for. Oops. Wash, rinse and repeat.

How deeps this fraud goes, is hard to ascertain. The other more sinister model is to hijack or hack a genuine vendor account with a good reputation, although I seriously doubt that this is the case in this particular instance. I will not be naming names in this article, as the latter scenario is still to be proven or disproved, and all the relevant evidence has been passed to the police concerning this trader, and I hope it will be investigated as criminal fraud.

What is really quite unpalatable about this whole matter is over a month down the road, Amazon is still mopping up. While the number of negative comments regarding this seller has clearly peaked, there are many upset customers still posting about this criminal. When you read the comments section, where disabled children saved up over months for a mobile phone or tablet, and are ripped off, it breaks your heart. While Amazon has stuck by their A-Z guarantee, that does not compensate for the disillusionment, nor the pain a parent has to experience in explaining to a child that the world is a big, bad place out there. My daughter is no innocent, but to use a more British and American colloquialism – Amazon has pissed on their chips. The phone in question was her 18th birthday present. And a disillusioned Millennial, a loyal customer does not make.

The problem here is a systemic one. 99% of the time, everything goes as expected, and it would be unfair of me to damn Amazon based on this one isolated incident. Overall, I have had an excellent experience shopping online with the company, although others who have tried to sell through their portal have expressed frustration with their lack of customer support. The biggest criticism seems to be their dependence on customer reviews as a selling tool, and that has been a clear refrain by consumer watchdog groups, as these metrics can be easily poisoned. The same applies to eBay *et al*, so it is not just Amazon that suffers from this problem. What is disturbing is their apparent complacency.

I raised my concerns with Amazon over three weeks ago regarding this trader, and while I have had a refund, the window of opportunity for these crooks has remained wide open, flapping in the wind. Meanwhile, the brand reputation of Amazon lies somewhere between Congress and colonoscopy, certainly as far as the additional hundreds of customers who have continued to voice their concerns and were ripped off in the meantime.

It is all very well selling convenience, but like a number of other major players in the digital economy, Amazon will now be faced with the unpalatable truth that reputation is slow to build and quick to erode. I have approached Amazon for comment, maybe I embarrassed them just a bit too much, by asking them the exact number of customers affected, and how much they have had to refund and what they intend to do about this whole sorry affair. After all, they are just as much a victim of this scam as my daughter. The best I could elicit from an Amazon spokesperson was the following statement:

“We are committed to providing our customers with the best possible shopping experience. All sellers on Amazon must adhere to our selling guidelines. Any seller found to contravene those guidelines will be subject to action from Amazon including removal of product listings and their account.”

“The Amazon A-to-z Guarantee provides additional protection for customers who buy from Amazon.co.uk’s third party sellers and if a customer received the item, but the item was defective, damaged, or not the item depicted in the seller's description, we will refund or replace that item. For more information on our A-to-z Guarantee, please visit:

http://www.amazon.co.uk/gp/help/customer/display.htmlie=UTF8&nodeId=3149571&qid=1239202264&sr=1-1”

I trust that Amazon does a lot more than just suspend these fraudsters’ accounts. I suspect we will never know the exact scale of any losses that Amazon have had to absorb, they may have some sort of payment escrow or re-insurance in place so their losses are in reality a lot less. Who knows the mysteries that can be covered by the magic words “commercially sensitive”? What is undisputed though, is irrespective of how much of a financial hit Amazon may have taken, this is an industry-wide problem, and unless the likes of Amazon, eBay, PayPal etc. get together with the banks, law enforcement and the couriers, fraud like this will continue on a growing scale.

Currently, this fraud looks as if it has burnt itself out. Apart from the occasional automated or prepaid 5-star rating that appears in a sea of condemnation, only a few unhappy customers have commented in the last few days, which is good news. The bad news is, I believe Amazon has hit an all-time record, with a vendor who has a 97% negative rating. Or if you prefer words to numbers, as one ripped off customer said: “They are pure scum, just avoid them.”

It appears that *Caveat Emptor* (buyer beware) is just as applicable now, as when it was first vocalized in English law in the 1600s, irrespective of how significant the brand is, or the modern technology.

Among clouds Performance and Reliability is critical

Download syslog-ng Premium Edition product evaluation here

Attend to a free logging tech webinar here

BalaBit IT Security

www.balabit.com

syslog-ng log server

The world's first High-Speed Reliable Logging™ technology

HIGH-SPEED RELIABLE LOGGING

- above 500 000 messages per second
- zero message loss due to the Reliable Log Transfer Protocol™
- trusted log transfer and storage

ServerU

Rack-mount networking server

Designed for BSD and Linux Systems

DESIGNED FOR FreeBSD | DESIGNED FOR GNU / Linux | DESIGNED FOR PRO apps FreeBSD Enterprise Appliance | DESIGNED FOR pfSense

Designed. Certified. Supported

Up to **5.5Gbit/s** routing power!

KEY FEATURES

- 6 NICs w/ Intel igb(4) driver w/ bypass
- Hand-picked server chipsets
- Netmap Ready (FreeBSD & pfSense)
- Up to 14 Gigabit expansion ports
- Up to 4x10GbE SFP+ expansion

PERFECT FOR

- BGP & OSPF routing
- Firewall & UTM Security Appliances
- Intrusion Detection & WAF
- CDN & Web Cache / Proxy
- E-mail Server & SMTP Filtering

contactus@serveru.us | www.serveru.us
8001 NW 64th St. Miami, LF 33166 | +1 (305) 421-9956