



®

freeBSD®

September/October 2022

JOURNAL

Security

- **Introduction to CARP**
- **Refactoring the Kernel Cryptographic Services Framework**
- **PAM Tricks and Tips**
- **SSH Tips and Tricks**
- **Pragmatic IPv6 (Part 3)**



FreeBSD[®] JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



2022 Editorial Calendar

- Software and System Management (January-February)
- ARM64 is Tier 1 (March-April)
- Disaster Recovery (May-June)
- Science, Systems, and FreeBSD (July-August)
- Security (September-October)
- Observability and Metrics (November-December)

Find out more at: freebsd.foundation/journal

Editorial Board

John Baldwin • Member of the FreeBSD Core Team and Chair of FreeBSD Journal Editorial Board

Tom Jones • FreeBSD Developer, Internet Engineer and Researcher at the University of Aberdeen

Ed Maste • Senior Director of Technology, FreeBSD Foundation and Member of the FreeBSD Core Team

Benedict Reuschling • FreeBSD Documentation Committer and Member of the FreeBSD Core Team

Mariusz Zaborski • FreeBSD Developer

Advisory Board

Anne Dickison • Marketing Director, FreeBSD Foundation

Justin Gibbs • Founder of the FreeBSD Foundation, President and Treasurer of the FreeBSD Foundation Board

Daichi Goto • Director at BSD Consulting Inc. (Tokyo)

Allan Jude • CTO at Klara Inc., the global FreeBSD Professional Services and Support company

Dru Lavigne • Author of *BSD Hacks* and *The Best of FreeBSD Basics*

Michael W Lucas • Author of more than 40 books including *Absolute FreeBSD*, the *FreeBSD Mastery* series, and *git commit murder*

Kirk McKusick • Lead author of *The Design and Implementation* book series

George Neville-Neil • Past President of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System*

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of AsiaBSDCon, and Assistant Professor at Tokyo Institute of Technology

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge

S&W PUBLISHING LLC

PO BOX 3757 CHAPEL HILL, NC 27515-3757

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsd.foundation.org

Copyright © 2021 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER

from the Foundation

Welcome to the latest security issue of the FreeBSD Journal.

As the Deputy Security Officer, I am especially interested in the security-focused issues of the Journal and this one does not disappoint. From an Introduction Networking Redundancy with CARP and CHERI Ports and Packages to Tips and Tricks for PAM and SSH, the issue provides an overview of technologies that can help keep your system secure. Members of the FreeBSD Security Team also sat down to talk about how the team works and why we do what we do.

Thank you to the volunteers who serve on the FreeBSD Security Team!

- Gordon Tetlow
- Baptiste Daroussin
- Xin Li
- Dag-Erling Smørgrav
- Glen Barber
- Ed Maste
- Mark Johnston
- Mariusz Zaborski
- Philip Paeps

As always, thanks to the FreeBSD Foundation, access to the FreeBSD Journal is free! Don't forget to share with your friends and colleagues and if you've a story idea for an upcoming issues, please let us know at jmaurer@freebsdjournal.com

Finally, thank you to everyone who contributed to the latest issue. We hope you enjoy it!

Ed Maste

Senior Director of Technology, FreeBSD Foundation and Member of the FreeBSD Core Team



Security

5 Introduction to CARP

By Mariusz Zaborski

11 Refactoring the Kernel Cryptographic Services Framework

By John Baldwin

21 PAM Tricks and Tips

By Michael W Lucas

27 SSH Tips and Tricks

By Benedict Reuschling

33 Pragmatic IPv6 (Part 3)

By Hiroki Sato

3 Foundation Letter

By Ed Maste

44 We Get Letters

Perplexed

By Michael W Lucas

47 Book Review: *Understanding Software Dynamics* by Richard L. Sites

Reviewed by Tom Jones

50 Interview: Keeping FreeBSD Secure

By Pam Baker and Anne Dickison

54 Conference Report: MCH 2022

By René Ladan

57 Events Calendar

By Anne Dickison

INTRODUCTION TO CARP

BY MARIUSZ ZABORSKI

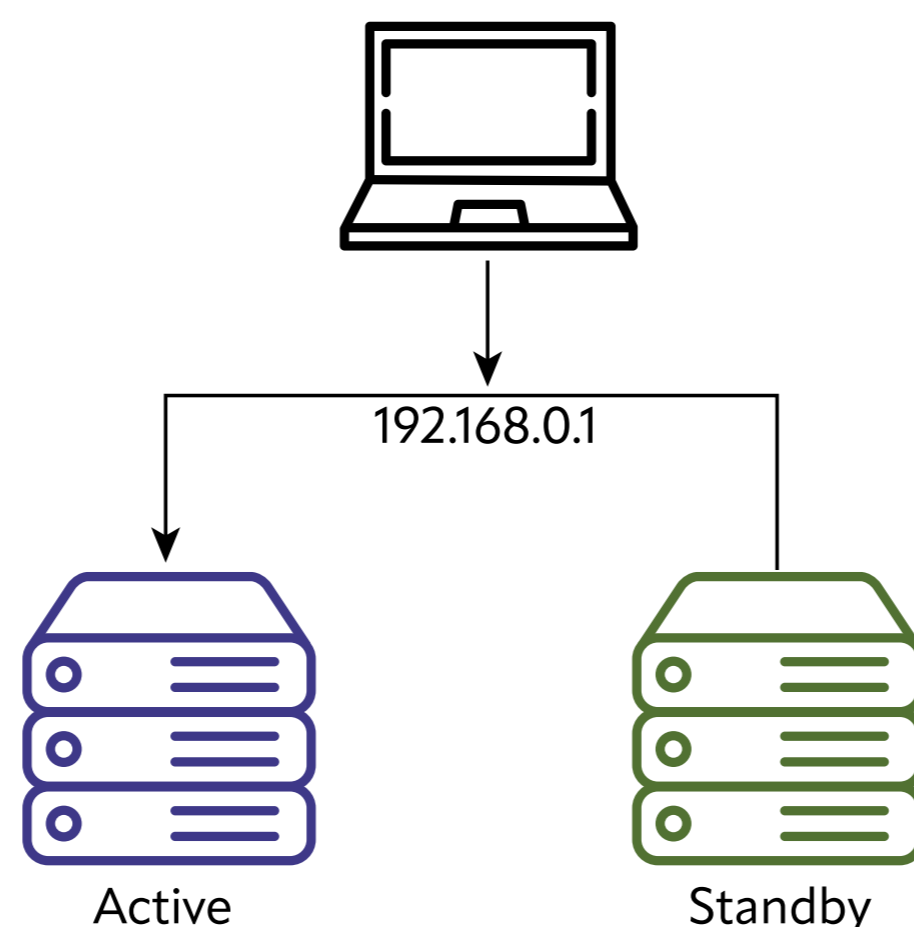
High availability topics might be challenging and complicated in large networks which is why we should look for solutions that are simple and easy to maintain and understand. CARP protocol, without any doubt, is one of them. CARP stands for *Common Address Redundancy Protocol* and its basic functionality is to allow multiple hosts to share a set of IP addresses.

The CARP protocol isn't new — it was first introduced in 2003 in OpenBSD as an alternative to CISCO protocol VRRP. CARP was to replace VRRP protocol because of patent issues, which are beyond the scope of this article. After CARP was introduced in OpenBSD, it was later integrated into FreeBSD and NetBSD. Finally, the ucarp was introduced — a userland implementation of CARP protocol — which brought an alternative to kernel implementations and made it available on Linux.

CARP Background

CARP allows a redundancy group — a set of hosts that share IP addresses. However, physically, only one interface has these IP addresses assigned (it is called the active host). In the case where an active host disappears (e.g., it was turned off or there is some issue with the network), other hosts in the redundancy group notice this and a new active host is elected. This situation is shown in Figure 1. There are two machines in the redundancy group — blue and green, but only the blue one is an active node, so other machines in the network don't have an issue choosing which to connect to.

Figure 1. Active and passive CARP nodes



In CARP, the active node is broadcasting its activity. This process is shown in Figure 2. Because the blue host is an active node, it is broadcasting CARP packages. The green and or-

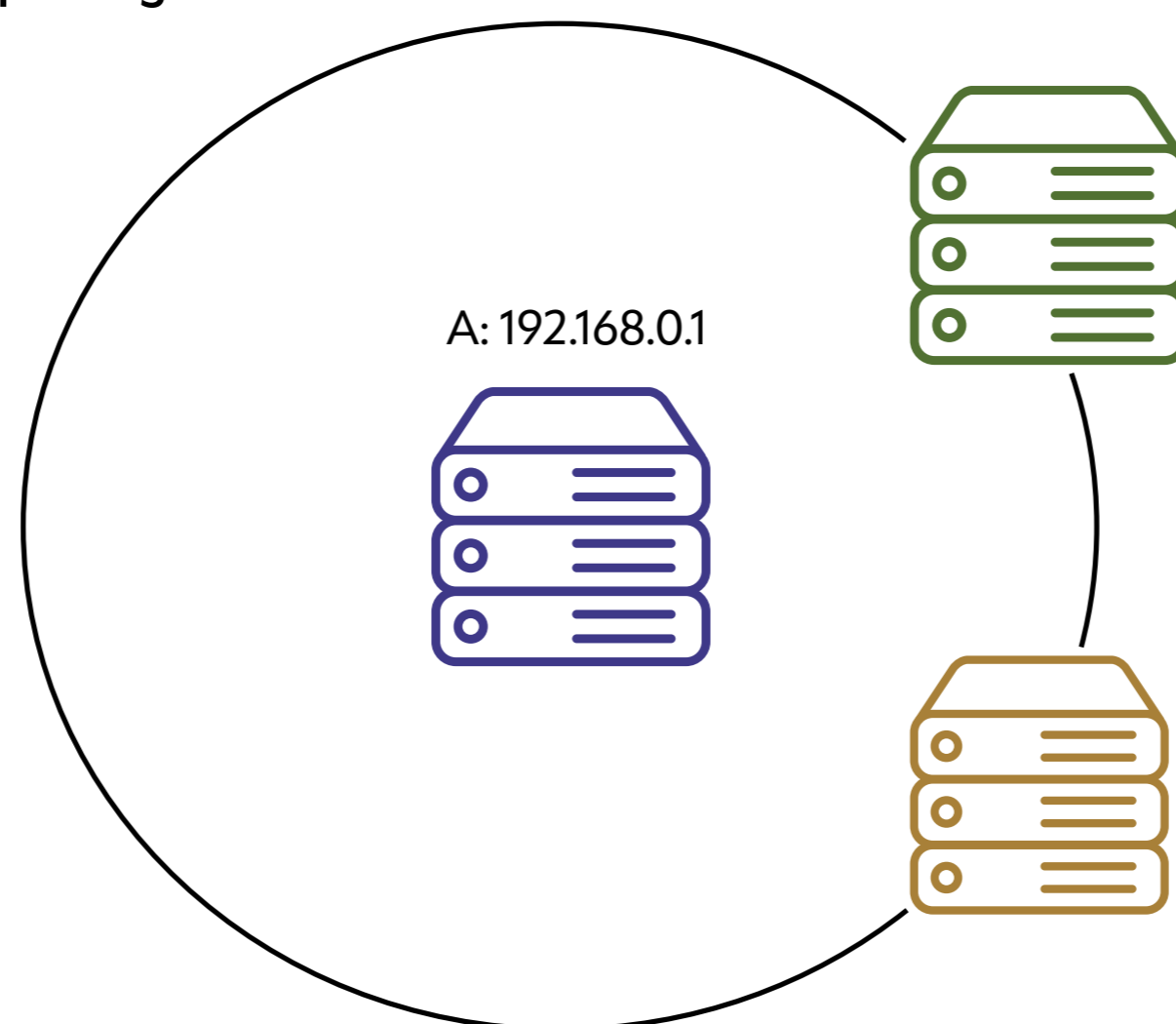
ange nodes are on standby and do not send any packages. The CARP package is quite small and contains only minimal information like:

- vhid (Virtual server ID), the identification of the redundancy group; all machines in the redundancy group have to share the same vhid
- Information about the CARP version and type of CARP package.

All packages in CARP are cryptographically signed, meaning each node in the redundancy group has to share the secret. CARP will never send its password in plaintext to the network. It is very important that every machine in the redundancy group be configured with exactly the same set of IP addresses. These IP addresses aren't sent over the network, however — they are used to calculate the cryptographical signature.

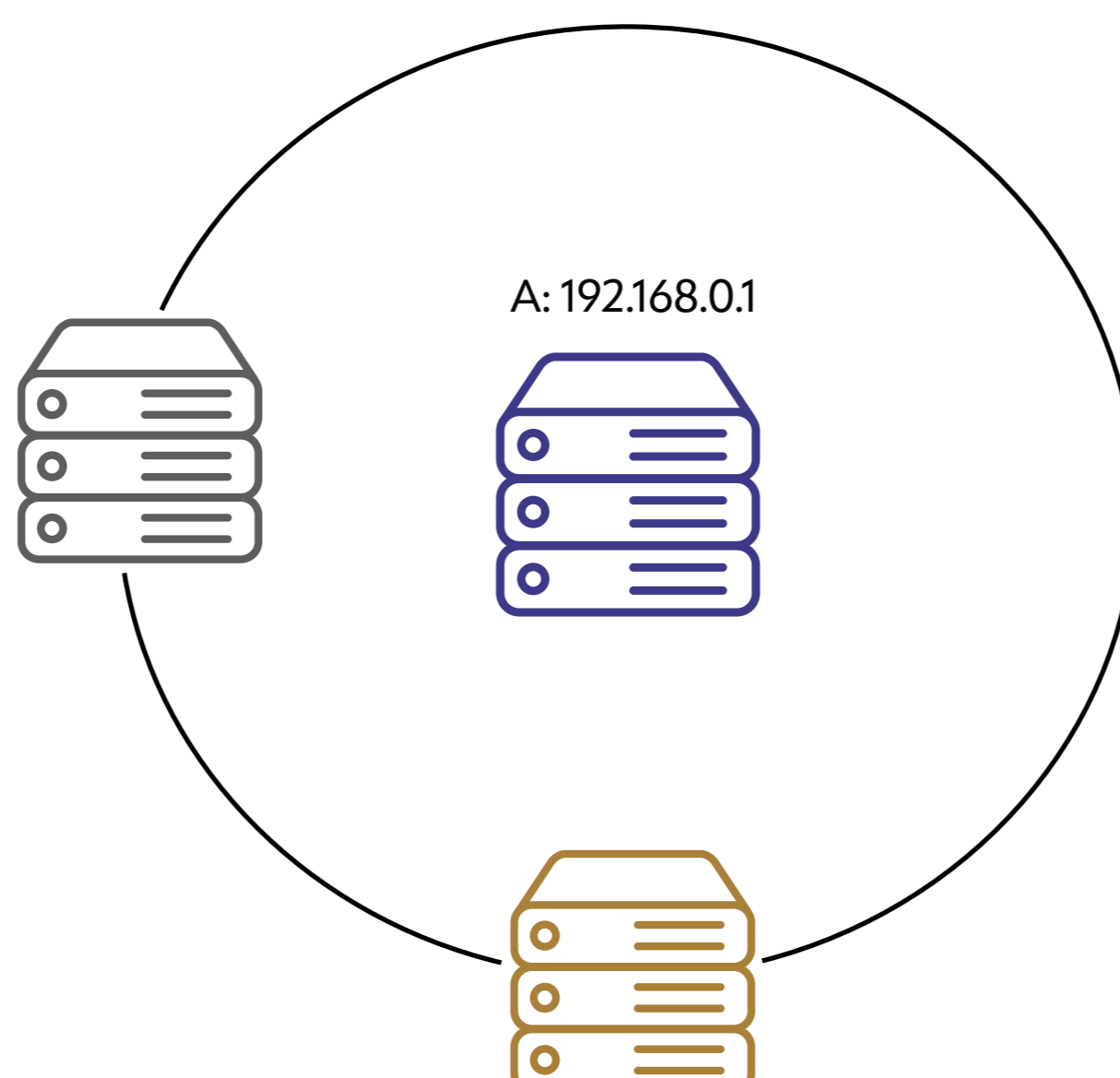
In the case shown in Figure 2, as long as the blue server is announcing cryptographically correctly signed packages with the given vhid, the other nodes don't do anything and just listen.

Figure 2. Announcing CARP packages



When a node stops receiving CARP packages for a while, another node decides to step in and become an active node. This situation is shown in Figure 3. The blue node, for some reason, stopped announcing the package, the green node noticed it, and now it has started to announce the CARP packages. When the blue node comes back, it will notice that the green node is now an active node and it will stay passive.

Figure 3. New active node



All examples above show a single IP address in the redundancy group. However, the redundancy group can have multiple IP addresses, and hosts can be in multiple redundancy groups — this is accomplished by different `vhid`. Thanks to that, we can also do some kind of load balancing among services in the network. For example, the green node can be an active node in the redundancy group, which provides the web server service, and the blue one can be an active node in the redundancy group, which provides the time service. If one of the nodes disappears, the other will become an active node in the other group.

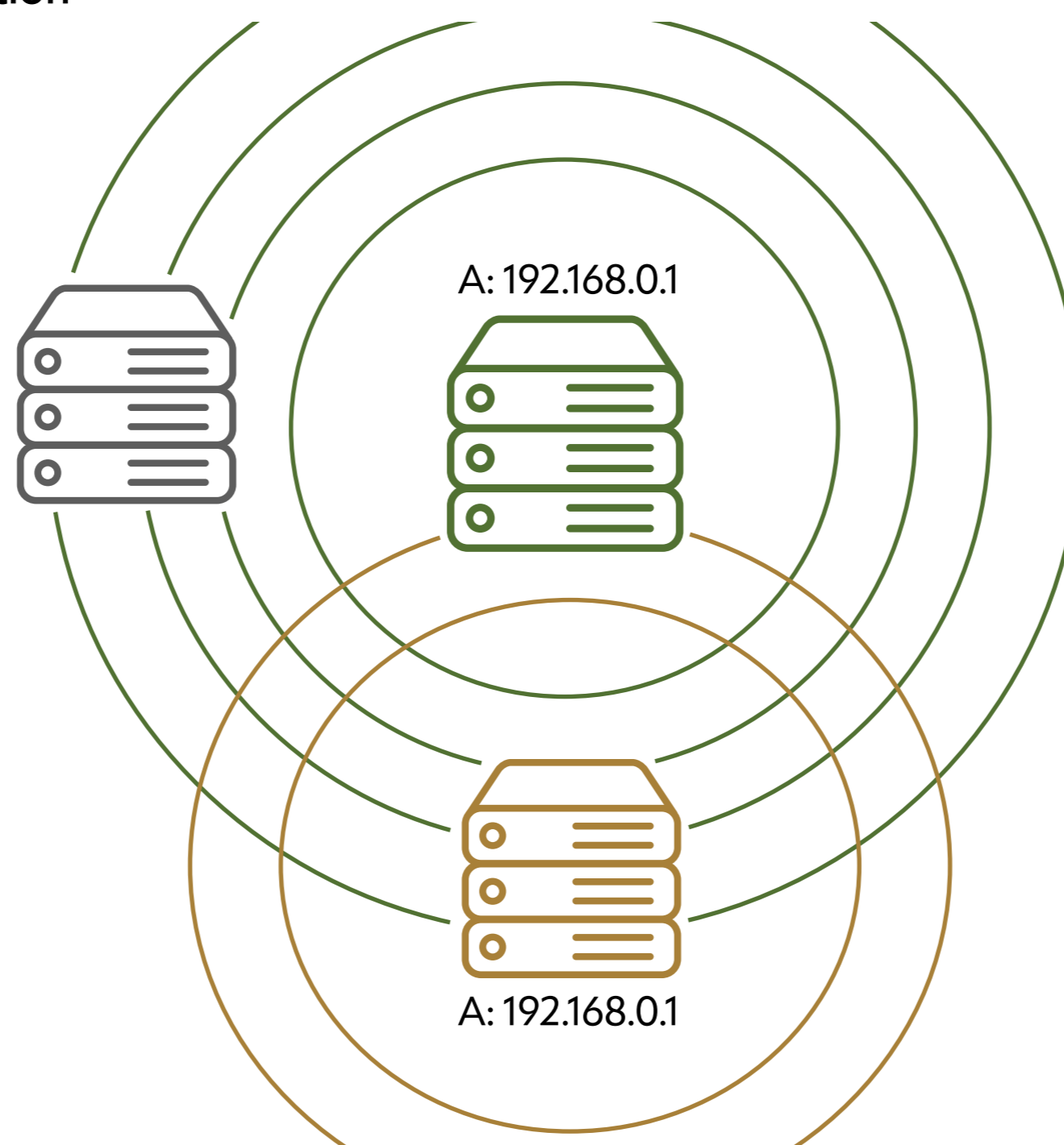
CARP and Split-brain

In a situation where two nodes notice, at the same time, that the node disappeared, both might want to become an active node. This is called a split-brain situation, where there are multiple active nodes. This situation might also occur when the link between the nodes is broken, and they stop seeing packages from each other and, after a while, the situation is fixed.

The split-brain issue is shown in Figure 4. CARP also solves this situation. When both hosts are active, both are announcing CARP packages. The node that announces more packages in a shorter period of time is the preferred node to become a new master. This is controlled in CARP with priority. Lower priority means packages are sent more often. When the other node sees that the CARP packages are announced more often than it is doing, it switches back to passive mode.

In the case when both nodes send packages with the same priority, the node will be chosen randomly.

Figure 4. Split-brain situation



FreeBSD Kernel Module CARP Configuration

CARP module is included in the default FreeBSD installation. From FreeBSD 10.0, CARP is no longer a pseudo-interface and it is configured directly on the interface. Listing 1 shows a basic configuration of CARP. First, we have to load a FreeBSD CARP module, which is accomplished by `kldload(8)` command. Then using `ifconfig(8)`, we define on which in-

terface the CARP should work (in our case it's `em0`). Next, we define a redundancy group ID (vhid is set to 1). Another important configuration is the passphrase used to calculate the checksum; this passphrase has to be shared among all hosts in the redundancy group. In the command, we also define the priority (or the advertisement interval). This is controlled by two parameters: `advbase` (advertisement base), which is specified in seconds, and `advskew` (advertisement skew — it is not shown on Listing) which is measured in 1/256 of a second. Just as a reminder — the lower priority means the host advertises more often, which means that it is a preferred node. Finally, we define which is the floating address.

On the same listing, we have two runs of `ifconfig(8)`; the parts not regarding CARP were omitted. In the first run, we can see that the redundancy group is in BACKUP state, which means that the interface is in standby mode and listening for CARP packages. Because there are no CARP packages in the network, it is switched to the **MASTER** (active) state, and the node starts to announce it. In Figure 5, we can see the captured CARP package, which is using a second static IP address for announcing the CARP packages to the multicast IP address. So, an additional IP address besides the shared one must be configured.

Listing 1. Configuration of CARP in FreeBSD

```
# kldload carp
# ifconfig em0 vhid 1 pass randompass advbase 1 alias 192.168.1.50/32
# ifconfig
em0:
    inet 192.168.1.50 netmask 0xffffffff broadcast 192.168.1.50 vhid 1
    carp: BACKUP vhid 1 advbase 1 advskew 0
# ifconfig
em0:
    inet 192.168.1.50 netmask 0xffffffff broadcast 192.168.1.50 vhid 1
    carp: MASTER vhid 1 advbase 1 advskew 0
    status: active
```

Figure 5. Captured CARP traffic using Wireshark

```
▶ Frame 3775: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface em0, id 0
▶ Ethernet II, Src: IETF-VRRP-VRID_01 (00:00:5e:00:01:01), Dst: IPv4mcast_12 (01:00:5e:00:00:12)
▶ Internet Protocol Version 4, Src: 192.168.1.157, Dst: 224.0.0.18
▼ Common Address Redundancy Protocol
  ▼ Version 2, Packet type 1 (Advertisement)
    0010 .... = CARP protocol version: 2
    .... 0001 = CARP packet type: Advertisement (1)
    Virtual Host ID: 1
    Advertisement Skew: 0
    Auth Len: 7
    Demotion indicator: 0
    Adver Int: 1
    Checksum: 0x04bd [correct]
    [Checksum Status: Good]
    Counter: 10009968722567644910
    HMAC: fb7f7879a6498338bee8bdf427c2b3c3a3c23fd0
```

What might come in handy is that FreeBSD `devd(8)` demon allows running additional scripts when the state has changed. Listing 2 shows an example of such a configuration from the FreeBSD man page. When the redundancy group changes its state, the `/root/carpcontrol.sh` script will be executed. The first parameter will be ``vhid@inet,`` and the second parameter will be the current state of the group.

Listing 2. devd(8) configuration for CARP

```

notify 0 {
    match "system"          "CARP";
    match "subsystem"       "[0-9]+@[0-9a-z.]+";
    match "type"            "(MASTER|BACKUP)";
    action "/root/carpcontrol.sh $subsystem $type";
};

```

ucarp

Additionally, a very promising project was a `ucarp`, the userland implementation of CARP protocol. It reduced the amount of code in the kernel space. Also, in the case of kernel space implementation, that might be slightly different. In this case, the code base was shared by multiple platforms. However, the project seems to have been abandoned--the GitHub project is closed, and the `ucarp` domain has expired. However, you can still find a `ucarp` distributed on different operating systems, so if you are looking for cross platform implementation, we still recommend you take a look at that project.

The configuration options are quite similar to the kernel implementations. Listing 1 shows how to install `ucarp` on a FreeBSD box. The next line shows its basic usage. Most options are self-explanatory at this point. Let's look into the `upscript` and `downscript` options. Because the `ucarp` was designed as a multiplatform tool, it doesn't know how to add an IP address to the interface — this responsibility was moved to the administrator. The user has to define his/her own scripts that add the IP addresses to the right interface.

Listing 3. Basic usage of `ucarp`

```

# pkg install carp
# ucarp --interface=eth0 --srcip=192.168.1.157 --vhid=1 --pass=randompass
--addr=192.168.1.50 --upscript=up.sh --downscript=down.sh

```

Another small caveat about `ucarp` is that we can define only one single floating IP address for the protocol. We can add many IP addresses in the `upscript` and `downscript`; however, only one (from parameter `addr`) will be added to the cryptographic signature. This means if we would like to mix the kernel and userland CARP implementations, it won't work with multiple floating addresses in the single redundancy group, because the checksum won't match.

Summary

Carp is a simple but very powerful tool that allows us to provide high availability in our network. There are two major CARP implementations: the kernel space (which each BSD operating system has) and one userland `ucarp` which is a cross-platform (and also works on Linux). Unfortunately, the userland implementation seems to have been abandoned. However, if you are looking for an easy and simple solution that will provide you with a floating address, you should still consider its use.

Bibliography

- CARP on Wikipedia — https://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol
- UCARP GitHub Project — <https://github.com/jedisct1/UCarp>
- CARP in FreeBSD Handbook — <https://docs.freebsd.org/en/books/handbook/advanced-networking/#carp>
- CARP FreeBSD man page — <https://www.freebsd.org/cgi/man.cgi?query=carp&sektion=4>

Acknowledgment

Figures in this article use resources from flaticon.com

MARIUSZ ZABORSKI currently works as a security expert at 4Prime. He has been the proud owner of the FreeBSD commit bit since 2015. His main areas of interest are OS security and low-level programming. In the past, Mariusz worked at Fudo Security, where he led a team developing the most advanced PAM solution in IT infrastructure. In 2018, he organized the Polish BSD user group. In his free time, Mariusz enjoys blogging at <https://oshogbo.vexillum.org>.



The FreeBSD Project is looking for

- Programmers • Testers
- Researchers • Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website

freebsd.org/projects/newbies.html

Downloading the Software

freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at freebsd.foundation.org

Help Create the Future. Join the FreeBSD Project!


FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by





Refactoring the Kernel Cryptographic Services Framework

BY JOHN BALDWIN


The FreeBSD kernel includes a cryptographic services framework used by other kernel subsystems for data encryption and authentication. Consumers of this framework include GELI, IPsec, kernel TLS offload, and ZFS. This framework is most commonly referred to by the acronym OCF. Originally, OCF stood for the OpenBSD Cryptographic Framework (<https://www.openbsd.org/papers/crypt-service.pdf>), but over time it has become an acronym for the OpenCrypto Framework.

History of OCF

The OpenCrypto framework was first imported into FreeBSD 5.0 as a port of the OpenBSD Cryptographic Framework by Sam Leffler in 2002. This initial version was primarily focused on supporting encryption and authentication of network packets for IPsec. It also included a character device driver, `/dev/crypto`, which supported custom `ioctl()` commands to permit userland applications to off-load cryptographic operations to cryptography co-processors.

OCF supported two different modes: symmetric and asymmetric. In symmetric mode, consumers (such as IPsec) first created a session describing parameters such as the algorithms to use and key lengths. Sessions were bound to crypto device drivers (either co-processor drivers or a software driver). Once a session was created, the consumer could issue one or more requests against a session. Each request performed a single operation such as encrypting or decrypting a buffer. Consumers explicitly destroyed a session once it was no longer needed. Asymmetric mode, however, worked differently. Rather than using sessions, each asymmetric operation was dispatched individually, and OCF chose a driver for each operation. Asymmetric operations were intended to assist with public-key cryptography and performed arithmetic on “big-numbers” such as computing a modulus according to the Chinese Remainder Theorem. Asymmetric operations were only used by user processes via `/dev/crypto`.

OCF supported two different modes: symmetric and asymmetric.



Symmetric sessions in OCF supported cipher and digest algorithms that were contemporary at the time. Supported ciphers included Cipher Block Chaining (CBC) modes of Data Encryption Standard (DES) and Triple-DES. Digest algorithms included Hash-based Message Authentication Codes (HMAC) constructions of MD5 and SHA-1. In addition, Encrypt-then-Authenticate (EtA) combinations of ciphers and HMACs were also supported.

Over time, a few additional algorithms were added such as SHA-2 digests and AES-XTS. However, the largest set of changes came in FreeBSD 11.0 with the addition of AES-CTR (a stream cipher) and AES-GCM an Authenticated Encryption with Associated Data (AEAD algorithm) by John-Mark Gurney. Modern versions of TLS and IPsec prefer AEAD algorithms and have deprecated other constructions such as EtA.

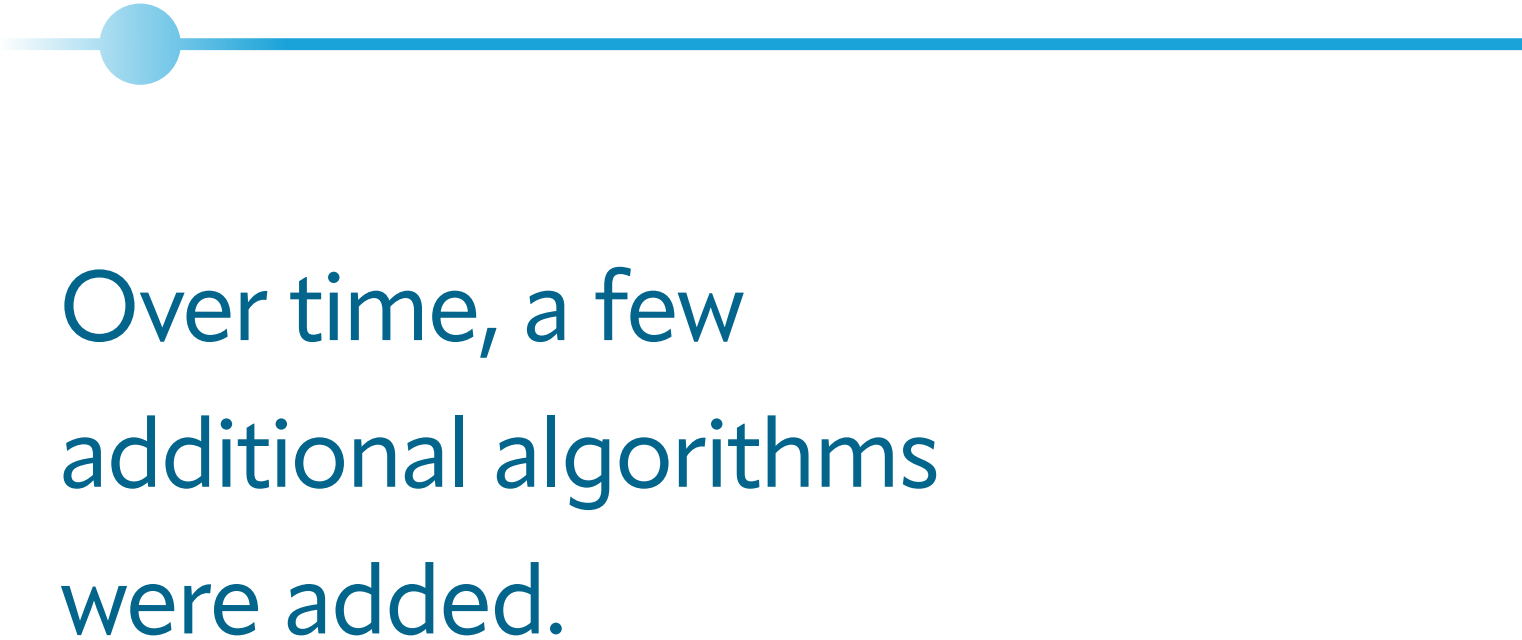
State of OCF in FreeBSD 11

During the 12.0 development cycle, I ported two Linux crypto drivers to FreeBSD (`ccr(4)` and an out-of-tree driver). This was my first experience with OCF, and I came away feeling that the interface had some quirks that resulted in extra “busy work” in crypto device drivers.


Linked-Lists for Transforms

Symmetric cryptographic operations in OCF are managed via sessions. OCF consumers create sessions describing the types of operations to be performed (algorithms to use, key sizes, etc.). Individual operations are then invoked on a session permitting drivers to cache state (such as precomputed key schedules) across operations. In 11.0, OCF session parameters were described by a linked-list of structures. Each structure defined either a single data transformation (such as symmetric encryption or compression) or a single digest computation. Sessions using a combination of algorithms (such as EtA) used separate structures for the encryption and authentication steps. Cryptographic operations also used a linked-list of crypto descriptors (one for each session parameter structure) describing the range of the data buffer on which to perform each operation as well as ancillary data such as keys and explicit Initialization Vectors (IVs) or nonces.

OCF in 11.0 did not define a specific order for session structures (e.g., encryption before authentication). Instead, consumers were free to construct the linked-list in an arbitrary order. As a result, drivers first walked the linked-list determining if there were unsupported combinations (e.g. multiple ciphers) as well as saving pointers to supported transforms (typically a cipher pointer and an auth pointer). Once this pass completed, drivers validated transform-specific parameters such as requested algorithms and key lengths. Operation descriptors were similar except that the order mattered. In particular, the `cryptosoft(4)` software driver depended on iterating over each descriptor in sequence to perform the desired set of operations. Other device drivers walked the descriptor chain validating that the chain contained the right number and type of descriptors (matching the session) again saving pointers to specific descriptors (e.g. cipher and auth descriptors) before completing the request.



Over time, a few additional algorithms were added.



Working with the linked-lists was not overly difficult, but it was tedious and resulted in a lot of code duplicated across drivers. Another common theme was that the OCF layer itself did not perform checks that were not driver-specific (such as validating the list of operation descriptors against the session). Instead, every driver was required to duplicate these checks. Similarly, OCF did not perform centralized checks on session parameters such as rejecting requests to create sessions with invalid parameters such as a cipher session with a key length that was not defined for the associated algorithm. These checks were instead duplicated across drivers.


AEAD Support

AEAD algorithms combine both encryption and authentication in a single algorithm. These algorithms provide similar functionality to EtA cipher suites but with a few changes. Notably, AEAD algorithms use a single key and nonce to provide both encryption and authentication. Under the covers, existing AEAD ciphers typically consist of separate encryption and authentication algorithms and derive separate keys and nonces for each from the user-supplied values. For example, AES-GCM uses AES-CTR for its cipher. AES-GCM is commonly used with a 12-byte nonce which is used as the upper 12 bytes of the IV with AES-CTR. The low 4 bytes are used as a counter which is incremented for each block of ciphertext. The keystream for block 1 is used as part of the computation of the authentication tag and the ciphertext is encrypted starting with the keystream for block 2.

In 11.0, OCF used separate session parameters and crypto descriptors for the cipher and authentication "sides". Consumers had to specify the same inputs such as keys and nonces for both sides, and drivers were required to check that both sides were consistent. In addition, due to a quirk of how some parts of OCF managed authentication algorithms, separate algorithm constants were defined for each key size (128, 192, and 256 bits) for the GMAC side of AES-GCM. Again, consumers had to ensure the GMAC constant matched the key size and drivers had to check for mismatches.

Crypto operations for AEAD in 11.0 also worked a bit differently than EtA requiring separate AEAD vs EtA logic in drivers. For EtA, OCF assumed that authentication was performed over a single region containing both auth-only data (such as the ESP header for IPsec) and the ciphertext. Furthermore, it assumed that these regions of the buffer were contiguous in the input data buffer such that they could be described by a single descriptor. This approach worked well with cryptosoft(4)'s implementation. AEAD algorithms, however, require more explicit separation of auth-only data (also known as Additional Associated Data (AAD)) from the ciphertext. AEAD algorithms define the order in which AAD is input into the authentication computation relative to the ciphertext regardless of the location within the associated data buffer. Existing AEAD algorithms also include the length of the AAD and ciphertext

Working with the linked-lists was not overly difficult, but it was tedious and resulted in a lot of code duplicated across drivers.



regions as inputs into the authentication function. To simplify the implementation of AEAD algorithms, the authentication descriptor for AEAD operations only covered the AAD region. It was implicit that the authentication algorithm must also be executed on the ciphertext region described by the cipher descriptor. Secondly, decryption operations for AEAD algorithms verified the supplied authentication digest, or tag, and failed the request with an error (**EBADMSG**) if it did not match. For EtA, OCF in 11.0 always computed the digest on the input and required the caller to save a copy of the original digest and to compare against the computed digest after the EtA operation completed.

IV/Nonce Handling

Crypto descriptors in OCF for cipher operations supported a tri-state of possible settings for dealing with IVs or nonces. First, an IV could either be supplied in a location in the data buffer (such as is commonly done in EtA algorithms for IPsec) or in a separate array in the descriptor. This choice was indicated by a flag in the descriptor. In the case that the IV was located in the data buffer, the driver would generate a random IV and insert it into the buffer for encryption operations unless the consumer set a second flag. In practice, none of the drivers in the tree supported generating IVs in hardware, so every driver duplicated the same block of code for managing IVs. This block of code had to check for invalid combinations of flags and, if the second flag wasn't set, call [arc4rand\(9\)](#) to generate a random IV to insert into the data buffer prior to encryption or authentication.


Session Handles

OCF sessions were named by integer IDs in 11.0. When a driver created a new session, it allocated a driver-private integer ID and returned it to the caller. This integer ID was supplied by the session consumer for each operation associated with a session as well as when removing a session. All existing drivers allocate driver-specific state for each session. When an operation is performed or a session is deleted, drivers use the session's integer ID to locate this driver-specific state. Existing drivers in the tree either used a $O(n)$ loop to locate the driver-specific state for each request, or they used a lock to protect access to a resizable array of pointers.

Session Probing

When an OCF consumer creates a new session, the OCF framework chooses a driver to service requests for the new session. In 11.0, this process was simplistic. Drivers registered a list of algorithms supported by OCF during their initialization. When a session was created, the framework would select a driver based on the requested algorithms. However, if the driver failed to create a new session because it did not support one of the other parameters (e.g., a key size, or if a device only supported standalone encryption or authentication but not combined EtA operations), OCF would propagate that failure back to the caller. Specifically, OCF would not try to fallback to another driver. For example, if OCF initially chose

When an OCF consumer creates a new session, the OCF framework chooses a driver to service requests for the new session.



a coprocessor driver that failed to handle the new session, the framework would not try to use a software driver which could handle the new session instead.

Streamlining OCF for Drivers and Consumers

As a driver author, OCF felt a bit clunky and required a fair amount of duplicated code among drivers. Some of this duplication was due to flexibility that did not seem needed. For example, using linked-lists for session parameters and operation descriptors permitted an arbitrary number and order of transformations. In practice, however, the operations used either required a single operation at a time, or a combination of one cipher and one authenticator such as EtA. On the other hand, OCF did not include flexibility that would be useful for some use cases. For example, KTLS did not always perform in-place encryption. To cater to OCF in 11 assuming in-place encryption, KTLS support via OCF had to copy the data into the output buffer before handing the data off to a crypto driver. KTLS also does not store its AAD inline in the on-wire format, but instead combines metadata about each TLS record along with some on-wire data from the TLS record to form the AAD.

To make OCF easier to work with, OCF has been refactored in the past couple of years. The primary goal of this refactoring has been to improve ease of use for driver authors. Improving performance is a secondary goal, and it is hoped that reducing complexity will have that benefit by leading to simpler drivers. All the changes described below shipped in FreeBSD 13.0, though the first change was made in 12.0.

Opaque Session Handles

The first refactoring was implemented in FreeBSD 12.0 by Conrad Meyer. Conrad replaced the integer session IDs used as a handle for OCF sessions with a new `crypto_session_t` opaque type. Under the hood these handles hold a pointer to a per-session structure allocated by the OCF framework. This per-session structure contains memory reserved for driver-specific session state. Drivers now provide the desired size for their driver-specific session state when registering with OCF. Before OCF asks a driver to initialize a new session, OCF allocates memory for the driver-specific session state. Drivers can obtain a pointer to this per-session state at any time via `crypto_driver_session()` as a cheap, $O(1)$ operation. When a session is freed, OCF zeroes and frees this driver-private structure as well. This removes the need for drivers to manage the lifecycle of driver-specific session structures using a model similar to that of `device_get_softc()` in new-bus.

Session Parameters

FreeBSD 13.0 introduced a new flat structure to describe symmetric cryptography sessions. This structure (`struct crypto_session_params` documented in [crypto_session\(9\)](#)) replaced the linked list of session structures and includes parameters such as key,

As a driver author,
OCF felt a bit clunky
and required a fair amount
of duplicated code among
drivers.

digest, and nonce lengths, session-wide keys, and a mode. Supported modes include stand-alone compression, encryption, and authentication as well as EtA and AEAD modes that combine both encryption and authentication. For device drivers the loop iterating over the linked list of session structures checking for multiple ciphers as well as identifying the cipher vs authentication structures has been removed. Instead, drivers can use switch statements on the mode and algorithm-specific fields (such as `csp_cipher_algorithm` and `csp_auth_algorithm`) to validate session structures and determine if a session is supported.

The parameter structure also includes room for future expansion. Including an explicit mode permits future combinations such as TLS's Mac-then-Encrypt (MtE) to be implemented if desired. In addition, the parameter structure includes a flags field to request optional features. During the initial conversion of drivers, all drivers were updated to reject sessions with a non-zero flags field. As new feature flags are added, drivers can opt-in to supporting sessions with those features by relaxing the checks on the flags field. If at least one driver supports each new feature that is added, this allows consumers to use new features without requiring changes in all crypto drivers. In practice this means that new optional features must be supported by the `cryptosoft(4)` driver.

Session Probing

FreeBSD 13.0 also introduced a new crypto driver method, `cryptodev_probesession`. When a new symmetric session is created, OCF invokes this new method on each eligible driver. Drivers can examine all the session parameters including the session mode and key lengths to determine if an individual session is supported. If a driver supports a session, it returns a bidding value from this method that OCF uses to choose the best-suited driver. Bidding values are similar to those used by `DEVICE_PROBE(9)` where a positive value indicates an error and the negative value closest to zero is considered the "best" driver. Unlike `DEVICE_PROBE(9)`, there are no special semantics for a return value of zero. Three bidding values are currently defined for coprocessor drivers, accelerated software drivers (such as `aesni(4)`), and plain software drivers.

Previously OCF did not provide a good way of distinguishing accelerated software drivers from coprocessor and plain software drivers. Prior to 13.0, accelerated software drivers were marked as coprocessor ("hardware") drivers to ensure they were preferred to plain software drivers. However, this also meant that userland requests submitted via `/dev/crypto` were enabled for accelerated software drivers by default. If userland software such as OpenSSL is going to use accelerated software instructions (such as AES-NI on x86), it is more efficient for userland to use those instructions directly rather than paying the additional overhead of system calls to encrypt or decrypt data. Userland requests via `/dev/crypto` only make sense when using a coprocessor (and even for many smaller requests the system call overhead can still outweigh the benefits of offloading operations to a coprocessor). The `cryptodev_probesession` hook provides preference for accelerated software drivers while avoiding conflating them with coprocessor drivers.

The parameter structure also includes room for future expansion.

Crypto Requests

FreeBSD 13.0 features a flattened crypto request structure (`struct cryptop` described in [crypto_request\(9\)](#)). This structure existed in older FreeBSD versions, but it no longer contains a pointer to a linked-list of descriptors. Instead, the information previously stored in descriptors such as the size and layout of regions in the data buffer such as AAD and payload are now described by members of the structure. Pointers to per-operation keys and separate IVs are stored directly in the structure as well. The new members in the structure assume that symmetric requests operate on a buffer containing AAD, IV, payload, and MAC regions. (Note that some regions are optional depending on the session parameters and request flags.) EtA modes are expected to apply authentication on both the AAD and encrypted payload regions while AEAD modes treat the AAD and payload regions as defined by the associated algorithm.

IV/nonce handling for requests has also been simplified. The OCF layer now generates any random nonces requested by a consumer before passing a request down to drivers. Drivers now only have to determine if the IV is stored inline in the data buffer or as a separate input in the request structure. A new helper function, `crypto_read_iv()`, permits drivers to fetch the IV from a request into a local buffer. This function eliminated duplicated code to read the IV from a request in almost all drivers.

IV/nonce handling for requests has also been simplified.

Crypto Buffers

FreeBSD 13.0 added additional abstractions for buffers holding data used as inputs and outputs of symmetric cryptographic requests. Prior to 13.0, crypto requests supported different types of data buffers including flat kernel buffers and `struct mbuf` chains. The type of buffer was encoded via flags in the `crp_flags` field of `struct cryptop` and an overloaded pointer pointed to the backing store. Two helper routines for moving data in and out of a crypto request's data buffer (`crypto_copydata()` and `crypto_copyback()`) accepted the flags field and overloaded pointer as arguments to support different data buffer types.

13.0 adds a new `struct crypto_buffer` type to describe a crypto data buffer. The structure includes an enum member which defines the type of the buffer as well as a union of type-specific fields. This permits buffer types which require more than a single pointer to describe. Using a dedicated type also permitted adding support for separate input and output buffers by storing two structures in `struct cryptop`. The existing `crypto_copydata()` and `crypto_copyback()` routines now accept the crypto request in place of the individual fields.

Two new API extensions further reduce duplicated code in drivers. First, new [bus_dma\(9\)](#) functions, `bus_dmamap_load_crp()` and `bus_dmamap_load_crp_buffer()`, permit loading a mapping for a crypto data buffer associated with a crypto request. This is primarily useful for coprocessor drivers which need to construct a DMA scatter/gather list to pass on to the coprocessor. Second, a cursor abstraction, primarily useful for software drivers, allows

drivers to iterate over virtual address ranges of a crypto data buffer. Cursors are bound to a crypto buffer when initialized. Drivers can then iterate over a data buffer either by copying data, which implicitly advances the cursor, or explicitly seeking forward. Logic specific to individual data buffer types is isolated in the implementation of crypto cursors rather than duplicated in software drivers. More details on the crypto cursor API can be found in [crypto buffer\(9\)](#). These extensions permit adding new data buffer types without modifying most existing drivers.

Finally, new helper routines have been added on the consumer side of the OCF API that are used to initialize the data buffer in a crypto request. Each crypto buffer data type has dedicated `crypto_use_*`() and `crypto_use_output_*`() routines that initialize a crypto request's data buffers. For example, `crypto_use_buf()` configures a crypto request to use a flat kernel data buffer as its input buffer. If a consumer does not specify a separate output buffer via one of `crypto_use_output_*`(), then the same data buffer is modified in place as both the input and output buffer.

Finally, new helper routines have been added on the consumer side of the OCF API.

Semantics Changes

Along with these structural changes, OCF in 13.0 also enforces several semantic changes. Some of these changes fall out from the structural changes while others are intentional towards the goal of simplifying drivers.

1. Sessions can now use at most one cipher and one authentication algorithm.
2. Sessions can only combine multiple algorithms in specific modes. For example, a session cannot mix compression and encryption.
3. Sessions can either use per-operation or per-session keys but not both.
4. Sessions which use per-operation keys instead of per-session keys must use the same key lengths for all operations.
5. AEAD sessions now use a single algorithm constant and key.
6. EtA sessions now validate checksums and fail operations with a bad MAC with `EBADMSG` similar to AEAD sessions.
7. Accelerated software drivers such as `aesni(4)` are now marked as software drivers instead of hardware drivers.

Existing consumers generally required only modest changes. Primarily these consisted of coping with structural changes such as using the session parameters structure. The only change that did not fall into this category was the change to validate MACs for EtA sessions. However, this generally simplified consumers by aligning code paths between AEAD and EtA sessions.

Driver Testing

The initial import of OCF included limited support for validating crypto drivers. The `tools/tools/crypto` subdirectory contained several utilities. Most of these fetched statistics for specific drivers or subsystems. One utility, `cryptotest.c`, did support some testing,

but it was primarily focused on measuring performance. For encryption algorithms it both encrypted and decrypted a random buffer and verified that the decryption result matched the original plaintext. However, it did not verify if the encrypted message matched a known-good standard. Similarly, for authentication algorithms this tool did no verification at all. It simply measured the performance of performing N operations.

Along with the changes to support AES-CTR and AES-GCM in 11.0, John-Mark Gurney added support for validating drivers against a set of Known Answer Tests (KAT) published by the [National Institute of Standards and Technology](#). The test vectors can be installed via the [security/nist-kat](#) port or package. The `test/sys/openssl/cryptotest.py` script is able to run these tests against crypto drivers and report any failures.

These two tests did have a few limitations. Both tests only supported a subset of algorithms supported by OCF. The KAT tests were an improvement over `cryptotest.c` since they validated encryption results against a trusted third party. However, the error reporting from the KAT tests was not detailed, and it was not easy to run an individual test against a driver when investigating a mismatch rather than the full battery of tests.

13.0 adds a new testing utility: `tools/tools/crypto/cryptocheck.c`. This utility uses OpenSSL's software cryptography as a gold standard to compare driver output against. This permits testing a broader range of algorithms. The `cryptocheck` utility also permits testing either individual operations or a set of operations spanning different sizes and/or algorithms. While the parameters such as keys and data buffers are populated with random data for each test, the userland RNG is not seeded so that the specific data inputs for individual tests are repeatable across multiple runs. Various parameters can be specified for tests including the sizes of plaintext buffers, keys, AAD, nonces, and MACs. For EtA and AEAD algorithms, `cryptocheck` also verifies that corrupted encryption buffers are detected and rejected with an error.

Documentation

The existing [crypto\(9\)](#) manual page has been updated and split into several pages. `crypto_session(9)` describes the session parameter structure and APIs to create and manage symmetric sessions. `crypto_request(9)` describes the symmetric crypto request structure and related APIs. `crypto_buffer(9)` describes crypto buffer cursors and other APIs that work on crypto request data buffers. [crypto_driver\(9\)](#) describes APIs for use by crypto drivers that are not described in one of the other pages. The [crypto\(7\)](#) page has been reformatted as a list of tables grouped by algorithm type and extended to cover all of the algorithms supported by OCF.

Various parameters can be specified for tests including the sizes of plaintext buffers, keys, AAD, nonces, and MACs.

Subsequent Changes

The set of changes above in 13.0 were authored by myself and mostly landed as a single [commit](#). Since then, OCF has been further extended by various developers.

Alan Somers [added](#) a new type of crypto data buffer that contains a list of VM pages. This permitted the use of unmapped I/O with GELI which improved performance by eliminating page table and TLB maintenance operations. Due to the abstractions around crypto data buffers, Alan's changes only touched a small number of crypto drivers directly. Most drivers worked with the new buffer type without requiring any changes.

I added support for [separate output buffers](#) and [separate AAD buffers](#) as new session feature flags. These improve the performance for kernel TLS by removing the need for data copies and for allocating temporary I/O vectors (struct iovec arrays). Since these requests were added as optional features, only drivers which wished to support kernel TLS needed to be updated to support these features.

Marcin Wojtas from Semihalf added another session feature to support [extended sequence numbers](#) (ESN) in IPsec for non-AEAD ciphers.

Support for additional AEAD ciphers have also been added. AES-CCM (used by OpenZFS) was included in 13.0. ChaCha20-Poly1305 (used by TLS and WireGuard) shipped in 13.1.

13.0 also removes support for older, deprecated ciphers and authenticators such as DES, TripleDES, Blowfish, and MD5-HMAC.

Conclusion

OCF still has lots of room for improvement, but the refactoring in 13.0 has succeeded in streamlining the API reducing code duplication and "busy" work in both drivers and consumers. (Mark Johnston told me that two OCF drivers he added in 13.0 were much easier to write due to the refactoring.) The changes sufficiently improved performance to permit kernel TLS to switch to using OCF instead of a private software crypto interface. The refactoring also provided a flexible base upon which other developers have been able to extend. I wish to thank Chelsio Communications and Netflix for sponsoring my OCF work in 13.0.

JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for over twenty years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger. John lives in Concord, California with his wife, Kimberly, and three children: Janelle, Evan, and Bella.

OCF still has lots of room for improvement, but the refactoring in 13.0 has succeeded in streamlining the API.

PAM TRICKS AND TIPS

BY MICHAEL W LUCAS

Pluggable Authentication Modules (PAM) are perhaps the least well understood, yet most broadly deployed part of FreeBSD. Every FreeBSD system uses PAM, via the OpenPAM suite. Most sysadmins don't touch it. If they have no choice but to change it, they follow cryptic online posts that look vaguely sensible. PAM is simpler than it looks, but its simplicity lets you accidentally achieve complexity. The big trick with PAM is to *not* do that.

In this article we'll look at PAM's components and configurations, help you avoid a few common mistakes, and figure out how to debug your mistakes.

What Is PAM?

Everybody agrees that usernames and passwords are not a great system of authentication. Nobody agrees on what should replace them. Different environments have different needs. Maybe you need Kerberos, or LDAP, or SSH certificates. Perhaps you authenticate sysadmins with hardware DNA scanners, or with bite guards molded to individual dental patterns. Each possible authentication method needs its own code.

You could try to compile every program to support every possible authentication method, but that leads to unsustainable code growth.

Or you could build a compatible shared library for each authentication method, and load that library only if you want to use that method. That's the "pluggable" part. Each library is an "authentication module," or just "module."

FreeBSD's OpenPAM, like primordial Solaris PAM, handles only authentication and authentication-related tasks. If you ever work with Linux, you'll see that the Linux-PAM developers decided that standard PAM was insufficiently complex and wedged other functions into their authentication stack. The features and tools found in OpenPAM work in most any PAM stack.

PAM Configuration

Configure PAM for FreeBSD's base system in `/etc/pam.d`, and PAM for packages in `/usr/local/etc/pam.d`. Each daemon has its own configuration file, named after the pro-

PAM is simpler than it looks, but its simplicity lets you accidentally achieve complexity.

PAM TRICKS AND TIPS

gram, that defines the PAM policy. Go look at the configuration for `sshd(8)`, `/etc/pam.d/sshd`. You'll find a bunch of lines like this.

```
.....
auth          sufficient      pam_opie.so          no_warn no_fake_prompts
auth          requisite        pam_opieaccess.so   no_warn allow_local
auth          required          pam_unix.so         no_warn try_first_pass
.....
```

Each line is one PAM rule. Each rule has four components: the type, the control, the module, and the module arguments. The first statement here has the type **auth**, the control **sufficient**, the module **pam_opie.so**, and the options **no_warn** and **no_fake_prompts**.

Rule Types

Authentication isn't only about the user credentials. The system must also check whether the access is permitted, provide resources for the user, and permit management of the authentication system itself. That's where the type statement comes in.

The *auth* type verifies the user's authentication information and sets resource limits. If you fat-finger your password or give a non-existent username, an auth rule kicks you out. The auth rules also establish limits like a maximum number of processes or amount of memory. We've seen auth rules above.

The *account* type controls access based on restrictions other than the user's authentication. If a user tries to log in out-of-hours, an account rule blocks that access.

The *session* type handles server-side setup. A command-line user needs a virtual terminal, a home directory, and probably a log entry saying they logged in. An anonymous FTP user should not get a virtual terminal and is limited to a particular directory. Session rules handle all this.

Finally, the *password* type handles authentication updates. Forced password changes are password rules.

Controls

You've seen access control lists in firewalls and server configurations. PAM rules are similar to access control lists. Each module can either reject, fail, or express "no opinion" on an authentication request. The control statement tells PAM how to process each module's decision. PAM has four common control statements: *required*, *requisite*, *optional*, and *sufficient*.

The *required* control means that this module must return success for the policy to permit access. If this module succeeds, the user gets access unless a later rule blocks it. PAM continues processing rules even after a required control fails.

A *requisite* control is much like a required control, but if an authentication module fails

Authentication isn't only
about the user credentials.

PAM TRICKS AND TIPS

rule processing stops immediately. This can leak information about why authentication failed.

Optional controls are used to support add-on functions like SSH agents or Kerberos. They can permit or deny access if and only if no other module rejects or accepts the session. You can use optional controls for features like adding time between a failed authentication attempt and the next attempt.

The *sufficient* control means that if this module succeeds and no previous required controls failed, the user gets access immediately. Rule processing stops. If the control fails, PAM does not deny access.

Modules and Arguments

PAM modules are shared libraries that implement specific authentication functionality. Passwords are a module. Checking LDAP is a module. Poking your SSH agent is a module. To understand a PAM policy, you need to know what each module does. Every OpenPAM module, and most add-on modules, have a man page. The first time you decipher a PAM policy, you'll read a whole bunch of man pages.

Each module can also have arguments. While each module can have its own arguments to handle its particular needs, most also accept a handful of common arguments. The *debug* flag logs extra information, to hopefully give you a chance to figure out why authentication isn't working as expected. The *no_warn* flag silences any user feedback on why the authentication request was rejected.

Password handling gets two special arguments, *try_first_pass* and *use_first_pass*. The *try_first_pass* option reuses any password the user already entered, but if that doesn't work, the module may prompt for another password. The *use_first_pass* option declares that the module should use the password that was already entered, and if that doesn't work, reject the request.

Reading A Policy

Let's consider the **sshd(8)** sample rules again. These are auth rules, so they involve accepting or rejecting login credentials. Consider the first rule.

```
auth                sufficient                pam_opie.so                no_warn no_fake_prompts
```

This rule is sufficient. If the module succeeds, the authentication request is granted, and rule processing stops immediately.

The module is **pam_opie.so**. The man page **pam_opie(8)** tells us this supports OPIE. You'll need to do a little more research to see that OPIE is One-time Passwords In Everything. This rule declares that if someone authenticates with OPIE, they get access immediately.

PAM modules are shared libraries that implement specific authentication functionality.

PAM TRICKS AND TIPS

```
auth requisite pam_opieaccess.so no_warn allow_local
```

This rule is requisite. If it fails, processing stops immediately. That seems... harsh?

This rule is for another OPIE module, `pam_opieaccess`. If the OPIE users were immediately granted access in the previous rule, why do we have another OPIE rule? A check of `pam_opieaccess(8)` reveals that this module checks to see if a user is configured to require OPIE. A user that requires OPIE, who correctly enters their OPIE information, should *never* hit this rule. They *should* be booted out.

Now look at the third rule.

```
auth required pam_unix.so no_warn try_first_pass
```

This is a required rule. A user who gets this far down must succeed with this module or be denied access.

The `pam_unix(8)` module checks the password file. It's the standard username and password authentication. So, this policy can be summarized as:

- If a user succeeds at OPIE, immediately let them in. Otherwise, keep going.
- If a user requires OPIE, reject them. A OPIE user with the right password will never get here.
- If the user enters a correct username and password, let them in.

Most accounts don't require OPIE, so we fall straight through to the password file.

That's very skimpy. What about users with invalid shells, or who have a shell of `nologin(8)`? What about users who aren't allowed to log in right now?

Those are account rules, not auth rules. If you check `/etc/pam.d/sshd`, you'll see a section of account rules specifically checking user access.

Adding Authentication Methods

People most often stumble into PAM when their organization starts requiring two-factor authentication. Maybe that's Google Authenticator, or a Yubikey, or Cisco's Duo. Perhaps you have specialized authentication hardware that reads voiceprints or fingerprints or aromaprints. We'll use the last three to build some less common authentication rules. Here's a fairly straightforward one. I haven't read the documentation for any of these nonexistent modules, so I'm going to ignore the options.

```
auth required pam_voice.so
auth required pam_finger.so
auth required pam_aroma.so
```

Most accounts don't require OPIE, so we fall straight through to the password file.

PAM TRICKS AND TIPS

All three policies are *required*. The user must submit a correct voiceprint and fingerprint, plus they must smell right, to authenticate.

```
auth    required    pam_voice.so
auth    requisite   pam_finger.so
auth    required    pam_aroma.so
```

The middle policy, for **pam_finger.so**, is requisite. If this policy fails, checking immediately stops and the application is informed of the failure. Performing aroma analysis is expensive, and we don't want to waste resources.

Perhaps you want to allow the user a choice of what authentication method to use.

```
auth    sufficient  pam_voice.so
auth    sufficient  pam_finger.so
auth    required    pam_aroma.so
```

Here, our first two authentication methods are sufficient. The user can use a voiceprint or a fingerprint. If those fail, the user must submit their stink. I could also make the last rule sufficient, but if the policy ends here, I'd want to add another rule invoking **pam_deny.so** to explicitly reject authentication.

PAM Debugging

Your carefully tuned policy doesn't work? Too bad. PAM provides very little explicit debugging. Your three choices are the debug argument, **pam_echo**, and **pam_exec**.

Many modules support a *debug* argument. This might feed debugging to the user. It could dump debugging to a log file. It could do nothing discernible. Modules ignore unsupported arguments, so adding debug arguments throughout your policy won't break PAM any worse.

The **pam_echo(8)** module takes a string of text as an argument. The string gets passed back to the user. These rules are always of type optional. Let's add some echo debugging to one of our experimental policies.

```
auth    optional    pam_echo.so "auth policy starting, trying voice"
auth    sufficient  pam_voice.so
auth    optional    pam_echo.so "voice failed, trying finger"
auth    sufficient  pam_finger.so
auth    optional    pam_echo.so "finger failed, taking a whiff"
auth    required    pam_aroma.so
auth    optional    pam_echo.so "how did we get here?"
```

If you think this looks like scattering **printf()** calls through your code, you'd be right. It was good enough for Sun in the 1990s so it's good enough for you.

If the program feeds the output back to the user, they'll get the debugging statements in their terminal.

PAM TRICKS AND TIPS

If the debug arguments don't provide useful information, and the program doesn't echo debugging output back to the user, then you get to get complicated with `pam_exec(8)`. The `pam_exec` module runs arbitrary commands for you. Yes, this means you could write a Perl script that checks user credentials against a Microsoft Excel spreadsheet over the network, but I hope you don't hate yourself *that* much. Most of the time, `pam_exec` is a way for intruders to mess with your authentication, but it's perfectly suited for calling a small shell script. Like this.

```
.....
auth    optional    pam_exec.so /usr/sbin/pamdebug.sh pam_voice
auth    sufficient  pam_voice.so
auth    optional    pam_exec.so /usr/sbin/pamdebug.sh pam_finger
auth    sufficient  pam_finger.so
auth    optional    pam_exec.so /usr/sbin/pamdebug.sh pam_aroma
auth    required    pam_aroma.so
auth    optional    pam_exec.so /usr/sbin/pamdebug.sh impossible_end_of_pam_rule
.....
```

The script itself is very simple.

```
.....
#!/bin/sh
logger "process $PPID calling $1"
.....
```

This logs the process ID and what stage of the authentication you're at. You wouldn't want to run this in a busy production environment where people are constantly logging on and off, but it makes debugging on a test system simple.

PAM has many more features and options than I can cover in this simple article, but hopefully you've gotten a decent idea of just how you can muck with your rules and fine-tune authentication to annoy your users in the exact manner desired.

Good luck.

MICHAEL W LUCAS is the author of *Absolute FreeBSD*, *FreeBSD Mastery: Jails*, and forty-eight other books. One of them was (drum roll please) *PAM Mastery*. Learn more at <https://mwl.io>.

SSH Tips and Tricks

BY BENEDICT REUSCHLING

SSH is a multipurpose tool used by pretty much anyone running and administering a Unix machine, logging in from one to the next more often than not. Secure, encrypted logins as SSH's core functionality are a great day-to-day help, but the daemon can do much more. Plenty of use cases are possible and I will only describe the ones that I use, which is by no means a complete list. I'll start with some basic hardening options and then move on to more sophisticated uses.

Login Options

SSH users that want to log into a system should use SSH keys instead of the keyboard-interactive authentication. Ideally, the latter authentication method (showing a password prompt) should be completely disabled to prevent any password script kiddie from constantly trying to break into poorly secured systems with weak user passwords. On some systems that I run for less tech-savvy users (I tried educating them on ssh keys, but to no avail), I still have to enable it. However sad this may be, I can still lock down the system to limit logins to a certain group of users and force some users to not use the keyboard-interactive method. Let's look at these one by one.

The SSH daemon is configured by the configuration file `/etc/ssh/sshd_config`. You can limit the users who are able to log in with the **AllowUsers** or **AllowGroups** directive. Even if a local user tries to log in and provides the correct password, as long as they are not listed in those directives, the login is still denied. In my use case, all the non-tech savvy usernames start with "abc", followed by their individual user ID. We can use wildcards to match on this username like this:

SSH users that want to log into a system should use SSH keys instead of the keyboard-interactive authentication.

```
AllowUsers abc*
```

Use `sshd -t` or even `-T` (for more tests) to check the validity of the `sshd_config` file before you restart the daemon to make these changes take effect. This line excludes the script kiddies from trying usernames like root, admin, and such. It's not completely secure but adds an extra hurdle to overcome. The **AllowGroups** directive does the same for a group of users.

User groups are easier to manage at a central location. Your new colleagues will certainly appreciate getting access immediately without having to visit your office first and beg to be let into that server. If they're added to the group, they can log in right away. Otherwise, you have to modify the **AllowUsers** line each time. The same is true when someone leaves, so do yourself a favor and use groups so as not to spend the rest of your days managing SSH access to a system that the whole company logs into.

Denying logins for certain users or even groups is possible with the **DenyUsers** or **DenyGroups** command, respectively. April Fools jokes aside, this is useful for restricting certain users in a group from accessing the system until they've returned the only existing keys to the server room (or are back from their vacations). The following order of directives is used when processing a mixture of such entries: **DenyUsers**, **AllowUsers**, **DenyGroups**, and then **AllowGroups**.

Restricting individual users' login methods is also possible using a match statement. This is comparable in function to an if statement and when such a match occurs, they override the global settings of **sshd_config**. Other match lines further down the file could undo settings made in previous match blocks, but let's keep this example simple for now.

The system described above where both keyboard-interactive and public key are used is monitored by a special user called monitoring. It periodically logs in with its own special SSH key, runs certain checks (disk full?), and reports the results back to the central monitoring server. This user should not be allowed to log in via the keyboard, since compromising this user would basically mean root access as some of the checks run with elevated privileges. That is why we tell the SSH server to only allow public key authentication when this user logs in. The match statement looks like this:

Restricting individual users' login methods is also possible using a match statement.

Match User monitoring

```
AuthenticationMethods publickey
```

Other users still use the global settings, but once the monitoring user comes along, the **AuthenticationMethods** setting gets overridden for this case. Other criteria for a match statement are **Group**, **Host**, **LocalAddress**, **LocalPort**, **RDomain** (the routing domain), and **Address**. Be careful here not to trust certain networks a user pretends to come from as this may be spoofed or rerouted. Find something that matches as little as possible to avoid long processing times or matching too many items, defeating the purpose of matching.

Note that not all keywords from **sshd_config** are changeable in a match statement, but many of them are. A full list is available in **sshd_config(5)**. Happy matchmaking!

Disconnecting Hung SSH Sessions

Has the following ever happened to you? You are logged in remotely on a server — doing some work — when suddenly your terminal freezes and you can't send any more commands to it? Or, you closed the lid of your laptop and opened it again at a different network location and can't get back to the session leaving you stuck at the terminal? I imagine you nodding in agreement, so here's why: it is because the server did not notice the network interrupt that may have happened and can't re-establish the connection. If this annoys you, check out the **net/mosh** package and see if that helps you.

How do you get the frozen shell back or at least properly disconnect? Even though it seems that there is no more communication happening between your SSH client and the server, there are still special commands that the server understands. Enter the following sequence of commands on the frozen terminal:

```
Enter ~ .
```

This sends a special interrupt command causing the server to immediately disconnect the session, returning control to your local shell session. Try it on a live session by hitting Enter, the tilde character followed by the dot. Be quick about it. Once it works, you'll quickly memorize this as a good practice if only to impress your co-workers with your knowledge.

Other sequences are documented in the ESCAPE CHARACTERS section of **ssh(1)**. Typing the sequence Enter, ~, and ? displays a list of escape sequences. Note that not all are supported by each SSH daemon, but in my experience, the disconnect works reliably.

The Hidden Login Script

Did you know that you can run a script each time a user successfully logs into a system via SSH? The file `/etc/ssh/sshrhc` that does this magic does not exist by default. When it is created and made executable, the SSH daemon will execute the commands listed in it. This happens after the environment files are read and right before the user's shell (or a command) is started.

Why would that be useful, you ask? It allows for custom initialization routines to run for this user. For example, a shell script could use the user's name for log in (available as **\$USER**) and ensure access permissions and ownership on the home directory are still restricted to that user. An error is echoed to **stderr** if the home directory does not exist for some reason. A simple script that does this may look like the following:

```
#!/bin/sh

HOMEDIR="/home/${USER}"

# Restore restrictive home directory permissions
if [ -d ${HOMEDIR} ]; then
    chmod 0700 ${HOMEDIR}
    chown ${USER}: ${HOMEDIR}
else
```



```
echo "Home directory ${HOMEDIR} does not exist" >&2
fi
```

Make sure the file is executable just like any other shell script to activate it the next time someone logs in. Other environment variables are available to use as well. Note that this runs every time a user logs in — even for file transfers via **scp/sftp**. Don't put in complicated code that takes a while to execute or the user will have to wait to finish the login process. For sneaky, behind-the-scenes actions, this is a good way as every user who successfully logs has to pass through the script.

Cheap, Yet Effective VPN

Virtual private networks have sprung up as paid services. They allow a secure connection between endpoints by tunneling the traffic and encrypting it over the internet. This helped people stay connected with the office during the pandemic, and even before that, when support personnel had to fix a server at ungodly hours of the day to prevent business interruptions. For Josephine random person, the aforementioned paid services enabled them to buy things cheaper by faking their origin connection to come from a different country. This could range from airplane tickets to as yet unreleased episodes of your favorite show on your favorite streaming service. While this may not yield success in all cases, it is nevertheless a convenient service to use.

How (and when) does SSH come into play? Well, each time we are on an untrusted, unencrypted network and we don't want prying eyes reading our traffic. This is often the case at train stations, airports, hotels, and libraries that offer free public WiFi. The connections there are often not (or not well) encrypted and shared between many different users--a perfect use case for an SSH-based VPN solution. It does not cost us anything since we have all the tools available. All that is needed is a publicly available host reachable on the internet that you can legitimately log in to.

Instead of directly connecting from our origin host *O* to the destination host *D*, we let our traffic take a little detour via host *P*. This takes care of giving the network packets a different origin address. Instead of your original host *O*, the packets will all be fetched by host *P* and forwarded to you via a SOCKS proxy. The destination *D* will communicate with host *P*, handling all requests, and each answer or result sent to *P* is forwarded to *O* in return. The SOCKS proxy will allow your browser to send and receive the packets, just like you'd browse a normal web page directly. It may be a bit slower than you are normally used to — because of the redirections between you and host *P* (the proxy) — but this is worth it to hide our origin address and the extra encryption you'll get — for free.



Virtual private networks
have sprung up as paid
services.

Here's How To Do It

Open a new terminal and type in the following, replacing the `sshproxyhost.example.com` with your SSH host the internet:

```
$ ssh -vD8080 -fCN sshproxyhost.example.com
```

This looks complicated, so let's explain each of the options provided:

- **v**: Gives SSH verbose output and may be omitted later once you're familiar with what's going on. At the beginning, it helps to debug the process and will emit any error messages that you wouldn't see normally.
- **D 8080**: This defines the local, dynamic port for the forwarding. On your local machine, the provided port (Note: 8080 in this case, a different one can be selected as long as it is unused by other daemons) is opened as a local socket with the other endpoint being the secure connection to the proxy server (creating it in the process).
- **f**: Puts the SSH process in the background so that the shell can still take other commands. Note that if you need to provide a password to log in, this will not work well. Generate an SSH key (using `ssh-keygen(1)`) for this connection and exchange it with the proxy host (`ssh-copy-id(1)`) for passwordless logins. This option is not strictly necessary, but useful once the whole process works.
- **C**: Compresses the encrypted VPN data. Depending on your network and processor speed, this may slow down or speed up the connection. Experiment with this option and remove it if it is too slow on the shabby hotel service where you're staying for one night only.
- **N**: SSH expects to run an interactive shell on the remote host, but we don't need this for our VPN. This option will not let SSH open a terminal and will only forward the ports-- which is what we want.

Each of these options is explained further in `ssh(1)`.

A typical session will emit similar messages to his one when using `-v`:

```
OpenSSH_8.6p1, LibreSSL 3.3.6
debug1: Reading configuration data /Users/bcr/.ssh/config
debug1: /Users/bcr/.ssh/config line 1: Applying options for *
debug1: /Users/bcr/.ssh/config line 16: Applying options for sshproxyhost.example.com
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 21: include /etc/ssh/ssh_config.d/* matched no files
debug1: /etc/ssh/ssh_config line 54: Applying options for *
debug1: Authenticator provider $SSH_SK_PROVIDER did not resolve; disabling
debug1: auto-mux: Trying existing master
debug1: Requesting forwarding of dynamic forward LOCALHOST:8080 -> *
debug1: mux_client_request_session: master session id: 7
...
debug1: Connection to port 8080 forwarding to socks port 0 requested.
debug1: channel 3: new [dynamic-tcpip]
...
debug1: channel 8: new [dynamic-tcpip]
```



```
debug1: channel 7: free: direct-tcpip: listening port 8080 for sshproxyhost.example.com port 443, connect from 127.0.0.1 port 58994 to 127.0.0.1 port 8080, nchannels 9
```

This confirms that the VPN is established. The line

```
debug1: Requesting forwarding of dynamic forward LOCALHOST:8080 -> *
```

shows how it works: you connect to your localhost port 8080. From there (follow the arrow), connections are established into the wide, wild world (www) of the internet. Replies are sent back in the reverse direction.

Your browser (or any other application that should use this VPN tunnel) simply needs to be set to use this SOCKS proxy in their connection settings. Look for an option like “manual proxy configuration”, set the socks host to “127.0.0.1” (localhost, see above), the port to 8080 (or the one you specified), and SOCKS5 proxy if there is such an option.

That’s it. Now, we should check if it works by browsing to a service like <https://www.whatismyip.com> (or similar sites) that display your public IP address. If this shows the IP of the host you used in your SSH command (sshproxyhost.example.com in my example), the VPN works. Wherever you next point your browser, the websites will establish connections, exchange traffic with this particular host and dutifully send you the traffic. Nice, isn’t it?

Some Words of Warning

As long as the SSH connection is open to the target system, the VPN tunnel is established. Be sure to re-establish the tunnel after your laptop goes to sleep as it may have disconnected you after some time of inactivity. If you’re renting a server on the internet to be a proxy for this purpose or someone else pays for the traffic on this system, don’t overdo it, as it may drive up costs. This is not a free solution in that case and if you use this often, you might as well pay for a professional VPN solution that gives you a couple servers across the world to choose from.

Also, be aware that you are not completely invisible. The SSH logs of the server used as a proxy will record your login information. Don’t do any malicious or harmful activities with this. We won’t send your next issue of the *FreeBSD Journal* to the prison planet that they put you on when they catch you.

I hope these tips and tricks were useful and will help in your day-to-day SSH interactions. Make sure to check out the man pages for both the client (**ssh(1)**) and the server (**sshd(8)**). For a more fun and comprehensive reading experience about all things SSH, I highly recommend the *SSH Mastery* book by Michael W. Lucas.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He’s also teaching a course “Unix for Developers” for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.

Pragmatic IPv6

(Part 3)

BY HIROKI SATO

The last column introduced typical examples of IPv6 deployment for a small network with a single uplink router, such as a network in your home. It did not cover some complex configurations involving DHCPv6 and/or PPPoE because we needed some more technical knowledge of IPv6 first. Before diving into such complex cases, let's learn more by configuring your FreeBSD box. This column will specifically cover the following two topics: what to do when you cannot get IPv6 Internet access from your ISP and how to configure IPv6 in essential utilities included in the FreeBSD base system.

Another Way To Get IPv6 Internet Access

The deployment scenarios in the last column assumed that your ISP offers an IPv6 service. In that case, the router between your ISP and your home network has an IPv6 GUA¹. All you have to do is a configuration of one or more IPv6 addresses and an IPv6 default router address pointing to the router.

So what to do if you get no IPv6 service? While the number of ISPs offering IPv6 service to their end-users is increasing, most are still optional and not their main service as of 2022. One of the ways to get access to the IPv6 Internet is the use of a tunneling connection. Figure 1 shows how "tunneling" works. Network A is an IPv6 network with IPv6 Internet access. Network B is an isolated IPv6 network. These two networks can be connected over the IPv4 Internet if they have an IPv4 address for the endpoints. Tunneling is a protocol translation technique that delivers a packet as a payload of another protocol. Namely, IPv6 packets can be delivered using IPv4 protocol, such as IPv4 TCP and IPv4 UDP. If you are familiar with the word VPN or virtual private network, you can understand tunneling as the technical foundation of VPN. Once the tunneling works, hosts on Network B can access the IPv6 Internet via Network A.

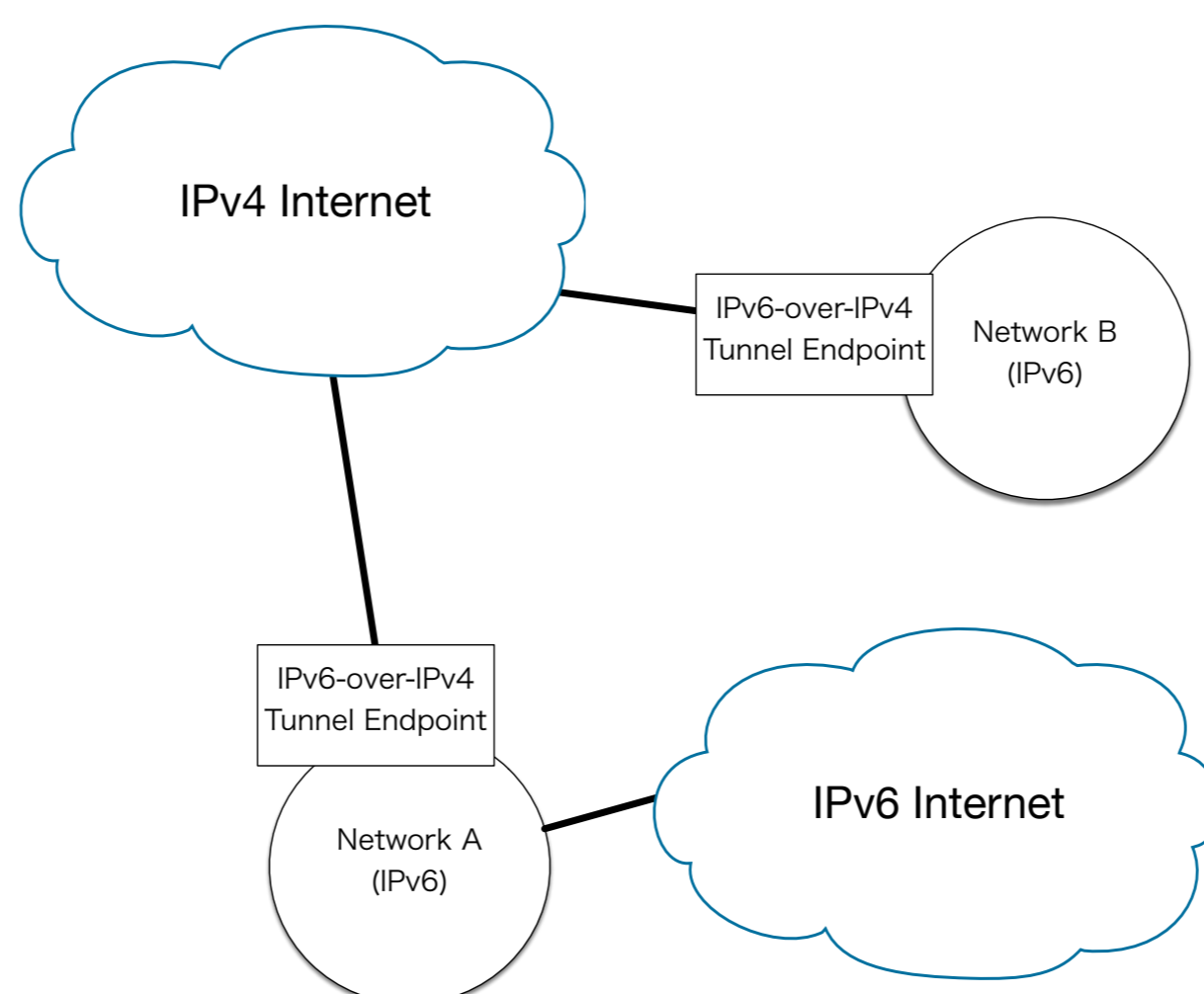


Figure 1: IPv6 networks connected over IPv4 communication

Several tunneling services support IPv6-over-IPv4 tunneling, which you can use for free. One of the reliable services, Hurricane Electric IPv6 Tunnel Broker, will be explained in the following subsections.

Hurricane Electric IPv6 Tunnel Broker

Hurricane Electric is a California-based Internet service provider focusing on Internet transit, data center colocation, and hosting services. Their network coverage is the most enormous scale in the world. It is usually unsuitable for a written article to introduce a specific Internet service like this because it rapidly becomes stale. Still, HE's IPv6 tunneling service has been maintained for over 20 years and been a good testing environment for people with no native IPv6 access. So the author recommends it as a way to get IPv6 Internet access for your experimental purpose. Although you cannot assume that it has the same reliability and performance as your native IPv4 Internet connection by your ISP, HE's service performs well, at least for personal use. And another advantage is that HE offers a /48 IPv6 address prefix. /48 is a recommended prefix length for ISPs so that their end-users can enjoy multiple LANs using the 16-bit². However, many ISPs offer only a /64. prefix or two.

Let's see how to configure the tunneling on your FreeBSD box.

Service Account Registration

The tunneling uses two endpoints on your network and in HE's network. Between them, a virtual network will be established over the IPv4 Internet. The endpoint on your side, a FreeBSD box, will act as an IPv6 router.

You must have a global IPv4 address on your FreeBSD box as a prerequisite. The tunneling uses IP packets with protocol number 41. This number is the same as IPv6, so the packet-filtering firewall is unlikely to block them. You must not use IPv4 network address translation by using the IPv4 private address space (10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16).

Visit <https://www.tunnelbroker.net/> and create your service account. After that, choose "create regular tunnel" in the web interface. Enter the IPv4 address of your endpoint, and select one of the HE's endpoints. All IPv6 packets from your network will be delivered to HE's network once before reaching the IPv6 Internet, so you should choose the nearest one. The author lives in Tokyo, so HE's Tokyo endpoint is the best, for example. By clicking the "create tunnel" button, your endpoint on HE's side will be configured. You need to click it twice because the first click checks if the endpoint works or not by sending IPv4 ping packets from HE's network. Ensure your firewall does not block the incoming ICMP echo request/reply. The IPv4 source address is shown in the list of HE's endpoints.

Network Parameters

Four IP addresses, "Server IPv4 Address", "Server IPv6 Address", "Client IPv4 Address", and "Client IPv6 Address" will be shown on the "Tunnel Details" page.

The Server IPv4 Address is the address for the endpoint on HE's side, and the Server IPv6 Address is for the IPv6 default router that should be used on your network. The Client IPv6 Address is for your endpoint.

Note that you must be aware of two networks — one between the two routers (this network is virtual) and your LAN. See Figure 2 for more details. A yellow circle means an interface. These two networks are separated by your router, which is the endpoint simultaneously in this configuration so that you will get two IPv6 prefixes. You can see the Server and Client IPv6 Address are on the same subnet prefix. That is the virtual network between the two endpoints. The prefix for your IPv6 LAN is shown as "Routed IPv6 Prefixes". You can

configure it to the LAN side on your router. By default, you will get a /64 prefix. By clicking "Routed /48" prefix button, you will get a /48 prefix in addition to it.

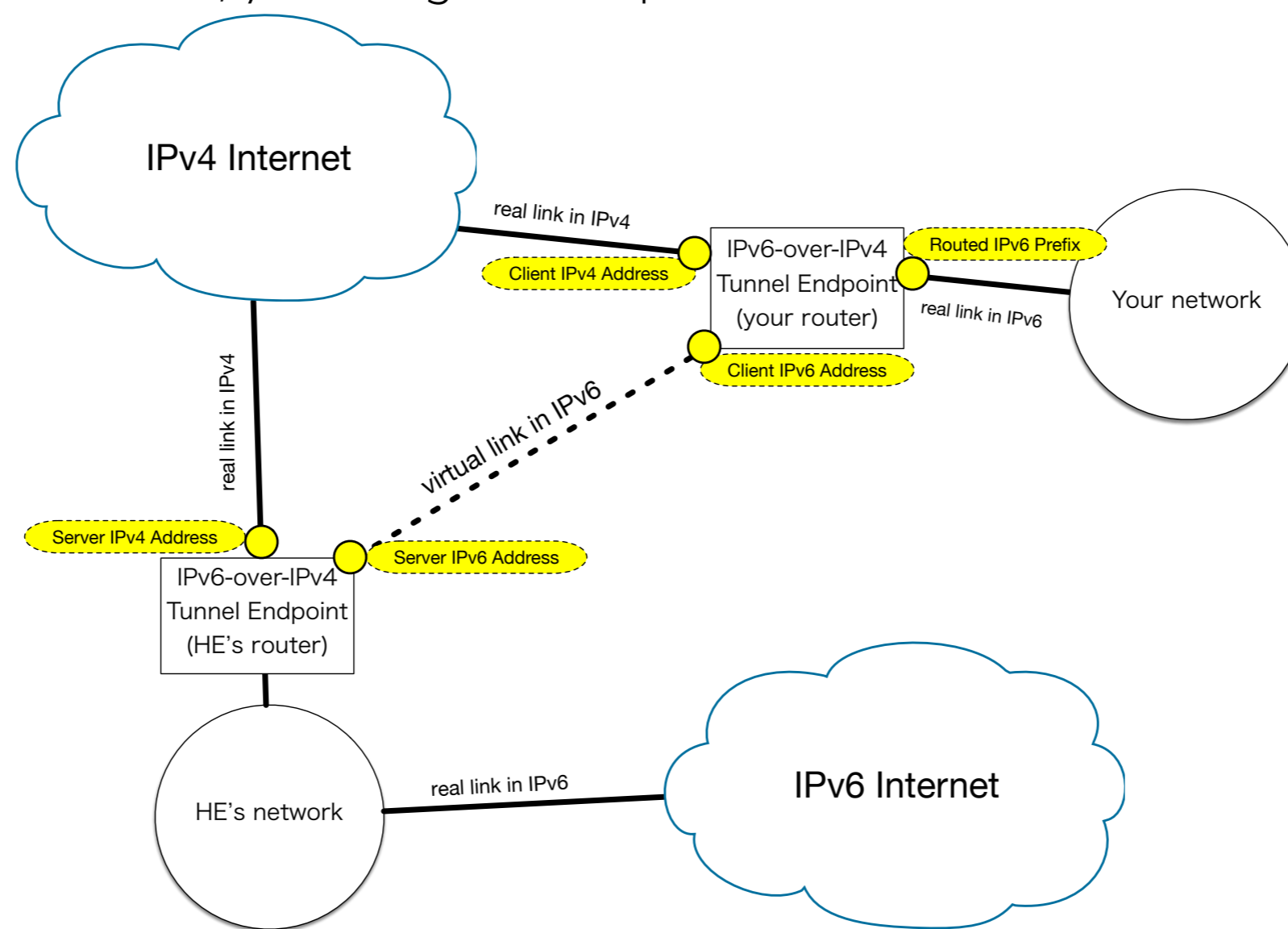


Figure 2: Tunneling between HE's network and your network

Now you are ready to configure your FreeBSD box.

Configuration on Your FreeBSD Box

The Tunnel Details page has "Example Configurations" tab and you can see a configuration example for FreeBSD 4.4 or later like this:

```
# ifconfig gif0 create
# ifconfig gif0 tunnel IPV4-CLIENT IPV4-SERVER
# ifconfig gif0 inet6 IPV6-CLIENT IPV6-SERVER prefixlen 128
# route -n add -inet6 default IPV6-SERVER
# ifconfig gif0 up
```

The keywords like IPV4-CLIENT mean the four parameters. This example is not wrong, but the following version is better:

```
# ifconfig gif0 create
# ifconfig gif0 inet tunnel IPV4-CLIENT IPV4-SERVER
# ifconfig gif0 up
# ping6 ff02::1%gif0
(hit Ctrl-C)
# ifconfig gif0 inet6 IPV6-CLIENT IPV6-SERVER prefixlen 128
# route -n add -inet6 default IPV6-SERVER
# ifconfig bge0 inet6 ROUTED-IPV6-PREFIX/64
# sysctl net.inet6.ip6.forwarding=1
```

The "gif0" is an instance of gif(4) pseudo-interface network driver on FreeBSD. An IPv6 packet is encapsulated in an IPv4 packet³ as shown in Figure 3. A rectangular represents a bit sequence from left to right. An IPv4 packet associated with a transport protocol, such as TCP or UDP, typically consists of an IPv4 header, a transport-layer header, and the payload (data). The three parts are concatenated to form a single packet and the packet is sent over

the network. In the same way, an IPv6 packet with the same transport protocol consists of the three parts. The difference is the first element.

If we consider an IPv6 packet as data for an IPv4 packet, we can send the IPv6 packet over IPv4 network. On the sender side, a router builds an IPv4 packet that has an IPv6 packet inside, and on the receiver side, another router extracts the IPv6 packet. This occurs on HE's router and your router when using `gif(4)` interface. In practice, encapsulation of an IPv6 packet into an IPv4 packet is done by prepending an IPv4 header to the IPv6 packet, as shown in Figure 3 because the transport-layer header is almost the same as each other.

To configure this interface, you need two sets of addresses. First, you have to create a new `gif(4)` interface by using `ifconfig gif0 create`. Then the tunneling can be configured by using "`tunnel`" keyword in the `ifconfig(8)` utility with two addresses for the endpoints. The first address is yours, and the second address is HE's.

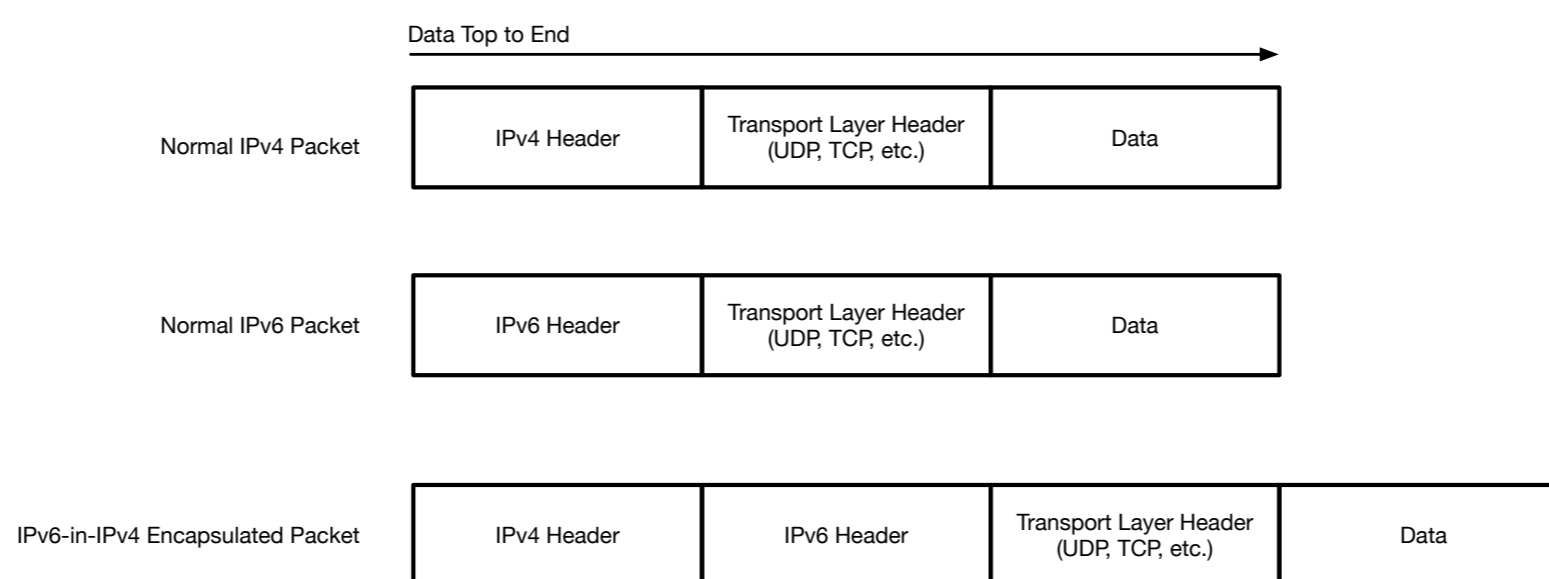


Figure 3: Encapsulation used in gif(4) interface

After that, a virtual network is established. Note that this virtual network has no connection state, such as "connected" or "disconnected". Packets are just sent as "IP datagram" between two IPv4 addresses on demand. An IPv6 packet is encapsulated, as shown in Figure 3, and sent as an IPv4 packet to the HE's endpoint, and it will be decapsulated in the HE's network. While the IPv4 packets may be lost somewhere on Internet, the error recovery is made in the upper-layer protocol, namely the IPv6 packet inside the IP datagram. Addresses for the tunneling are often called "addresses for the outer protocol". The outer protocol, in this case, is IPv4.

To check if the virtual network works, you can send an IPv6 ping. After the second line of the example configuration, enter `ifconfig gif0 up` and try an IPv6 ping by using the automatically-configured LLA⁴ of the `gif0`:

```
# ifconfig gif0 create
# ifconfig gif0 tunnel IPV4-CLIENT IPV4-SERVER
# ifconfig gif0 up
# ping6 ff02::1%gif0
PING6(56=40+8+8 bytes) fe80::80c8:4a75:123a:8a32%gif0 --> ff02::1%gif0
16 bytes from fe80::80c8:4a75:123a:8a32%gif0, icmp_seq=0 hlim=64 time=0.084 ms
16 bytes from fe80::4a52:2e06%gif0, icmp_seq=0 hlim=64 time=4.765 ms(DUP!)
16 bytes from fe80::80c8:4a75:123a:8a32%gif0, icmp_seq=1 hlim=64 time=0.081 ms
16 bytes from fe80::4a52:2e06%gif0, icmp_seq=1 hlim=64 time=2.738 ms(DUP!)
^C
--- ff02::1%gif0 ping6 statistics ---
2 packets transmitted, 2 packets received, +2 duplicates, 0.0% packet loss
round-trip min/avg/max/std-dev = 0.081/1.917/4.765/1.970 ms
```

The virtual network is working fine if you get the (**DUP!**) response. This is because you should have a reply from the router on the HE's side and another from the **gif0** interface itself. If you get no response from another side of the endpoint, double-check the configured IPv4 addresses and packet filters located between the two.

Then you need to configure actual IPv6 addresses for communication. This can be done using the **ifconfig(8)** utility, the sixth line of the example configuration. This line configures a point-to-point network that has only two IPv6 nodes. At this point, you can send an IPv6 ping to *IPV6-SERVER*.

The above configuration is sufficient if you just want access to the IPv6 Internet from the box. Your FreeBSD box can now work as an IPv6 host node. To make it a router for your /64 and /48 LANs, you have to configure the default IPv6 route to *IPV6-SERVER* and enable the IPv6 packet forwarding feature. The following part in the example does it:

```
# route -n add -inet6 default IPV6-SERVER
# ifconfig bge0 inet6 ROUTED-IPV6-PREFIX/64
# sysctl net.inet6.ip6.forwarding=1
```

This configuration assumes that the interface facing your LAN is **bge0**. *ROUTED-IPV6-PREFIX* is your prefix assigned by HE's service. You should have both /64 and /48. You can configure both on the same interface or on a different interface.

There is one thing you need to think about here. What IID⁵ should be used? That is the router's address on the LAN.

You can see that IPv6 addresses used for the tunneling interface were **::1** and **::2** in their IIDs. That is one of the strategies. However, the author recommends using the all-zero address for simplicity. If you get **2001:db8::/48** from HE, pick up a /64 network in that range, say **2001:db8:1::/64**, and configure it as the router address. The IID is all-zero. This looks scary because an all-zero host address in IPv4 has a special meaning. Although it has never been clearly defined in RFC or other specifications, some traditional network implementations have recognized it as a broadcast address, not a unicast one. While the all-zero host address works with no problem on FreeBSD, the author guesses that you have avoided it in practice if you learned TCP/IP network in 20th century. In IPv6, the all-zero IID is completely valid. There is another reason why the all-zero address is suitable for a router, but it will be covered in the later columns.

After configuring the default router and the router address on the LAN, you can configure the hosts on the LAN. See also the last column about what options you can take. Running **rtadvd(8)** on **bge0** to enable automatic configuration is highly recommended.

Configuration in */etc/rc.conf*

Once you confirm if your IPv6 tunnel works, put the configurations by hand into */etc/rc.conf*:

```
cloned_interfaces="gif0"
ifconfig_gif0="inet tunnel IPV4-CLIENT IPV4-SERVER"
ifconfig_gif0_ipv6="inet6 IPV6-CLIENT IPV6-SERVER prefixlen 128"
ipv6_defaultrouter="IPV6-SERVER"
ipv6_gateway_enable="YES"
ifconfig_bge0="inet6 ROUTED-IPV6-PREFIX/64"
```

Before rebooting, you can check if it works by using **service(8)** command like this. Two ping6 are to check the tunnel endpoint reachability and the default router configuration:

```
# service routing start
# service netif restart gif0
# ping6 ff02::1%gif0
# ping6 IPV6-SERVER
```

That's all. You can now build IPv6-capable Internet servers on your LAN because the configured IPv6 addresses are reachable from the IPv6 Internet.

There is one additional comment about **ipv6_defaultrouter**. While the above example uses *IPV6-SERVER*, it can also be the router's LLA on HE's side, you can see by sending a ping, or you can set it to **"-interface gif0"**. The former is not recommended because the LLA can change when HE changes its equipment. It is recommended to use an LLA for a static route only when it is manually configured and under your control. The latter is another way to simplify the configuration. Because **gif0** is configured as a point-to-point interface, the route to the router on the HE's side can be specified by using the interface name on your side. Doing this allows you to use a consistent configuration even if the address IPV6-SERVER is changed.

Using IPv6 in Essential Utilities

Another topic in this column is practical configuration examples for software in the FreeBSD base system. After you obtain access to the IPv6 Internet, let's make software use of IPv6.

General Rules and Pitfalls

From the user's point of view, the most significant difference is the notation of an address. Except for that, TCP and UDP work just like IPv4. So you can get started by replacing an IPv4 address in a configuration file with one in IPv6. The first column covered OpenSSH as an example. Let's see what change is required.

Configurations for the **sshd(8)** daemon is stored in **/etc/ssh/sshd_config**. You can edit it directly, but modifying the configuration using command-line flags allows you to keep the default configuration file intact. For example, you can put the following lines into **/etc/rc.conf** to change some of the configurations partially:

```
sshd_enable="YES"
sshd_flags=" \
-oPort=22 \
-oUsePAM=no \
"
```

Let's go back to the IPv6 configuration topic. The **sshd(8)** daemon listens to **tcp/22** in both IPv4 and IPv6 by default. You can see the following lines in **/etc/ssh/sshd_config**. All lines are commented out, but it means they are enabled by default:

```
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
```

"::" is an IPv6 address. This all-zero address is called "unspecified IPv6 address". Note that this is the all-zero subnet prefix and the all-zero IID simultaneously, not the same as all-zero IIDs in the previous section. This particular IPv6 address means any of IPv6 addresses. So the **sshd(8)** daemon listens to all of the IPv6 addresses configured on the box.

Like this, some software simply adopt raw IPv6 addresses in the configuration file. In that case, you can write an IPv6 address instead of IPv4 without further consideration. The **sshd(8)** is smart enough to distinguish which address family is used. If you want to restrict IPv6 addresses for the **sshd(8)** daemon, you can put a **-oListenAddress** option:

```
sshd_enable="YES"
sshd_flags=" \
-oPort=22 \
-oUsePAM=no \
-oListenAddress=2001:db8::1 \
"
```

Although IPv6 addresses in a configuration file should be written in the recommended way in RFC 5952⁶, almost all of software accept a redundant notation such as **2001:0db8:0000:0000:0000:0001**. And if it is an LLA, you must add **%zoneid** part.

However, some do not use raw IPv6 addresses because they break the backward compatibility with IPv4. Let's see **syslogd(8)** as an example:

```
syslogd_enable="YES"
syslogd_flags="-s -cc -b [fe80::f4:a6ff:fe43:50b%epair5b]"
```

This is one of the configurations used on the author's FreeBSD box. The **syslogd(8)** daemon has a **-b** option to choose the listening address and port number. It listens to the address and accepts incoming UDP packets as a logging information source. The address format in IPv4 is **"-b address:service"**. The *service* is a port number or a service name listed in **/etc/services**. Thus the **syslogd(8)** daemon cannot recognize which colon is for *service* part if a raw IPv6 address is used.

To solve this issue, the **syslogd(8)** daemon uses a format like **"[ipv6-address]:service"**. An IPv6 address must be enclosed with square brackets. If there is the **%zoneid** part, it must be inside the brackets. This is a popular format among software using **"address:port"**.

Although this square-bracket format works if the software supports it, you have to be aware of the brackets are one of the meta-characters of shell programs such as **/bin/sh**. Actually, the above example for **syslogd(8)** does not work because the square brackets are sometimes interpreted as meta-characters. A pair of brackets will be replaced with a file or directory name if the string matches. It works if there is no matched name, but if there is any, the configuration fails strangely. The following is another working example:

```
syslogd_enable="YES"
syslogd_flags=" \
-b 192.168.0.10 \
-a 192.168.0.0/24 \
-a [fe80::ffff:2:200%bridge0]:* \
"
```

```
-b [fe80::ffff:1:202%bridge0] \
-a [fe80::%bridge0]/64 \
"
```

You can see "*" as an argument of the `-a` option. This means the daemon accepts any UDP port. This may match one or more filenames and be replaced when the command line is evaluated. You might consider that this can be solved using "\" just before "*". It may or may not work because it depends on how the shell variable is evaluated. So you must be careful about the pathname expansion when software requires the square-bracket format. Of course, there is no problem when it appears in a configuration file.

Another point where you should pay attention is how to specify a prefix length by using a format like "/64". In the above example, the prefix length is outside the brackets. This is because the prefix length is not a part of an IPv6 address. Locations of a raw address, the zone id part, and square brackets are sometimes confusing. Remember that `[fe80::/64%bridge0]` and `[fe80::%bridge0/64]` are all wrong.

The last pitfall is a case when specifying an address by using a hostname. You can often put a hostname where an address is supposed to be specified. The hostname is usually resolved by name services available on the system. A single name can have multiple addresses because DNS supports and may return multiple A and AAAA resource records. Under such a situation, some software needs clarification about what address and which address family will be used. There is no consistent way to specify an address family of a hostname. You should have a unique hostname with a single address if you want to use hostnames.

Let's see more working examples.

DNS nameserver in `/etc/resolv.conf`

For simplicity, the configuration of recursive DNS servers should be handled by RA messages explained in the last column. However, if you have a DNS server on the same link, you can add an LLA manually and specify it like this:

```
nameserver fe80::ffff:1:35%bge0
```

One of the reasons why using a manually-configured LLA is that you can use the same address on different networks. It dramatically simplifies the administration.

However, LLAs in `/etc/resolv.conf` do not work for software that depends on LDNS library and does not use resolver functions in FreeBSD libc. This means that the `drill(1)` utility does not work with LLAs while there is no problem with using IPv6 GUA in `/etc/resolve.conf`. And `dhclient(8)` does not work, either. As a workaround, you need "nameserver" line by using IPv4 address or IPv6 GUA⁷.

NFS and `/etc/exports`

The `/etc/exports` file uses raw IPv6 address format. It supports the prefix length notation for "network" keyword. An example is as follows:

```
/a/ftproot -alldirs -maproot=0:0 -network 2001:db8:1::/64
/a/ftproot -to -alldirs -maproot=nobody:nobody -network fe80::%lagg0/10
```

Note that NFS does not fully support LLAs because the RPC library always handles IPv6 addresses without the zone ID. The second line can be specified, and it is a correct notation,

but it does not work, unfortunately⁸. IPv6 GUA works fine. Avoid to use LLA for NFS for now. It should be fixed on FreeBSD 14.

Sendmail

The sendmail program also uses the raw IPv6 address format. And it supports the address family selection keyword for each address. An example is as follows:

```
sendmail_enable="YES"
sendmail_flags="-L sm-mta -bd -q30m \
  -ODaemonPortOptions=Family=inet,address=127.0.0.1,Name=MTA \
  -ODaemonPortOptions=Family=inet6,address>:::1,Name=MTA6,M=0 \
  -ODaemonPortOptions=Family=inet,address=0.0.0.0,Port=587,Name=MSA,M=E \
  -ODaemonPortOptions=Family=inet6,address>:::,Port=587,Name=MSA6,M=0 \
  -ODaemonPortOptions=Family=inet,address=0.0.0.0,Port=465,Name=MSA,M=s \
  -ODaemonPortOptions=Family=inet6,address>:::,Port=465,Name=MSA6,M=s \
"
```

You can safely use a hostname with both a single IPv4 and a single IPv6 address simultaneously because of "Family=" keyword.

Configuration of the transport used by MSA⁹ is handled by lines in `/etc/mail/FreeBSD.submit.mc`:

```
dn1 If you use IPv6 only , change [127.0.0.1] to [IPv6:::1]
FEATURE('msp', '[127.0.0.1]')dn1
```

In this configuration file, a modified square-bracket format is used because the sendmail program requires the square-bracket format even for IPv4. If you want to use IPv6, `[IPv6:raw-ipv6-address]` must be used.

Syslogd

The command-line options were explained in the previous section. In the configuration file, `/etc/syslog.conf`, a remote host can be specified by using the raw IPv6 address format. This is an example used to receive logs from a jail environment running ISC BIND:

```
+fe80::e:1ff:fec5:e80b%bridge100
!-named
*.notice;authpriv.none;kern.debug;lpr.info;mail.crit;news.err; /var/log/ns/messages
!named
*. * /var/log/ns/named.log
!*
+@
```

Summary

Getting access to the IPv6 Internet by using tunneling service and configuring essential software included in the FreeBSD base system to use IPv6 are explained.

Tunneling using `gif(4)` interface is not specific to Hurricane Electric IPv6 Tunnel Broker. It can be used to build your own virtual network; while it may be too primitive for a practical

purpose — it has no support for encryption or automatic configuration. FreeBSD provides a rich set of tools for network experiments.

While IPv6 configuration for userland programs typically involves only writing IPv6 addresses into the configuration file instead of IPv4, several pitfalls are listed in this column. There is also software that does not handle an IPv6 LLA correctly. Try IPv6 on your box, and if you find a problem, please report it to the author and/or the FreeBSD project.

In the next issue, more software configuration and NDP (Neighbor Discovery Protocol), one of the IPv6 core protocols you should be familiar with will be explained.

Footnotes

¹ GUA stands for “global unicast address”, which is routable in the IPv6 Internet.

² Remember that an IPv6 GUA has the prefix and the IID, and the IID is almost always 64-bit long. If you have a 48-bit prefix, you can have 65,536 networks in your LAN by choosing the 16-bit. If you have a 64-bit prefix, you can have a single network only.

³ This protocol is defined in RFC 4213, “Basic Transition Mechanisms for IPv6 Hosts and Routers”.

⁴ LLA stands for Link-Local Address. After `ifconfig gif0 up`, an LLA is automatically configured. See also the last column for more details.

⁵ IID stands for Interface IDentifier. The lower bits in an IPv6 address that is unique for each host. The length is usually set to 64, but theoretically, any length equal to or shorter than 128 – (subnet prefix length) is allowed.

⁶ Rules for text representation of an IPv6 address are covered in the first column. See also RFC 5952: “A Recommendation for IPv6 Address Text Representation”.

⁷ The author is working on these problems, and it will hopefully be fixed in near future.

⁸ The author is also working on these problems.

⁹ MSA stands for Mail Submission Agent. This is a program to submit an email to an MTA, Mail Transfer Agent. The `sendmail` program can be used as an MSA or an MTA.

HIROKI SATO is an assistant professor at Tokyo Institute of Technology. His research topics include transistor-level integrated circuit design, analog signal processing, embedded systems, computer network, and software technology in general. He was one of the FreeBSD core team members from 2006 to 2022, has been a FreeBSD Foundation board member since 2008, and has hosted AsiaBSDCon, an international conference on BSD-derived operating systems in Asia, since 2007.

Support FreeBSD®



Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsd.foundation.org/donate



WeGetletters

by Michael W Lucas



Dear Journal Letters Column,

I have to integrate this new hardware doohickey into all our authentication systems on all our hosts, no matter which operating system they're using. It's harder than I thought. The differences between OpenSolaris and FreeBSD and Linux and AIX and HP/UX and all the other Unixes are all tiny — but taken together they seem huge. Is there an easier way to do this?

—Perplexed

Perplexed,

I recently had the chance to go to my first concert in three years—Nine Inch Nails, Nitzer Ebb, and Ministry. I kept myself safe, with my stick-on mask and ear plugs and eye goggles and full-body bunny suit, not to mention the barbed wire halo, but at least I was able to attend this glorious outpouring of incendiary rage and righteous betrayal and the kind of defiant bitterness that gives me reason to crawl out of my cage and scrape the bile off my teeth every morning.

That approaches how I feel about PAM.

“But you have to have PAM” people shout. “It’s a necessary evil, it’s a standard!” Nope. It isn’t. It looks like a standard so long as you don’t look at it. Sun Microsystems, the well-spring of NFSv2 and Java and many other seductive immortal nightmares, offered it up to the public in the hope it would be adopted. It was. Sun did not organize an Interop as they did for NFS or maintain Java-style control. Instead, they left everybody free to implement it in their own preferred, slightly different manner. Yes, yes, the Common Desktop Environment became a standard back in the 1990s and mentioned PAM integration, but any standards that coexisted with Saturday morning cartoons and the ankylosaurus should not be considered relevant today. The closest thing we have to a PAM standard is in the document *X/Open Single Sign-on Service (XSSO) — Pluggable Authentication Modules* from an attempt to nail it into POSIX, but folks followed about ninety percent of that, and we all know that ninety percent compatible equals zero percent interoperable.

We don’t even have a standard language. Is it a PAM policy or a chain? Rules or modules? Types or rules? Even if you read the documentation, you can only follow it through intuition and good karma.

The Journal’s editors saw fit to have some PAM apologist write a piece for this issue. I won’t glorify it by calling it an “article,” because he probably cut-and-pasted snippets of his book for it and shamelessly ended it with a plug for same. I’m not saying that he’d do

anything for a buck, but if I was unlucky enough to be near him, I would absolutely mention in the sort of voice I normally reserve for screaming back at Al Jourgensen that “ethics” are a thing even in information technology and that he’s doing all this in public where anybody who exerts a morsel of effort can figure out his little scam. Fortunately for him, nobody cares enough about his feeble antics to bother.

Forget standardization. Not everything *has* to be a standard — otherwise, we couldn’t make the mistake of inventing new things. Look at how PAM works. You grab these shared libraries, never mind where they came from or how carefully they’ve been audited, chain them together, and force them to collectively decide how your authentication is going to work? We all know how access control lists work. *This* is allowed. *That* is not. You carefully define the characteristics of permitted activity and block everything else. What you do not do is implement a wishy-washy system where rules can say things like “yes, but only if everybody else agrees” or “I’m gonna veto it, but y’all go ahead and vote.”

Voting? Security is not a democracy! It’s not even a republic.

Not everything has to be a standard — otherwise, we couldn’t make the mistake of inventing new things.

Not that PAM holds a proper vote. It’s more like a bunch of drunk programmers deciding what to order for dinner. You go around the table, sure, but finally the one with the deepest understanding of compiler internals picks whatever will give everyone the worst hangover possible. The others get to pick a couple of side dishes and maybe ask for a pack of fortune cookies, even though the cashier keeps reminding everyone that they do fortune saganaki because they’re a Greek joint and *your fate is always delicious*.

How is that access control, especially without wonton soup?

Fine. Fine. Here we are.

But another thing—debugging. I fully understand that all debugging boils down to scattering print statements throughout the code and watching it go wildly astray, but PAM doesn’t even have a standard way to do that. Maybe debug statements will work. Perhaps you can use PAM’s “echo” module and spit stuff back at the user, which will absolutely never terrify that guy from Shipping & Receiving who needs three tries and divine intervention to successfully log onto the menu-based inventory system. He’ll be fine. Pinky swear.

So you use `pam_exec` and write a little script that dumps information to a log file, or maybe even into `logger(1)` and straight into the system log. Using a shell script in your authentication system doesn’t guarantee you’ll get broken into, especially if they’re extremely simple, but shell scripts have this horrid tendency to grow and every line of code is a vulnerability. You might as well write a little Perl script that checks authentication credentials against a Microsoft Excel spreadsheet over the network.

Wait—the PAM apologist already suggested doing exactly that?

Time to lower my standards. Again.

But, again, here we are. PAM is the standard that isn’t. We’re stuck with it.

The only consolation I can offer is that your impressions are valid. Nothing is compatible. Everything uses its own language. I’m told that the Pope declared that time spent

configuring PAM counts as time served in Purgatory, however, so be sure to fill out your time sheet correctly.

Hope? Yes, I have hope. I hope is that systemd swallows Linux-PAM and OpenPAM becomes the Last Stack Standing. Perhaps then we can have an authentication system designed by sober people who know how to order fortune cookies.

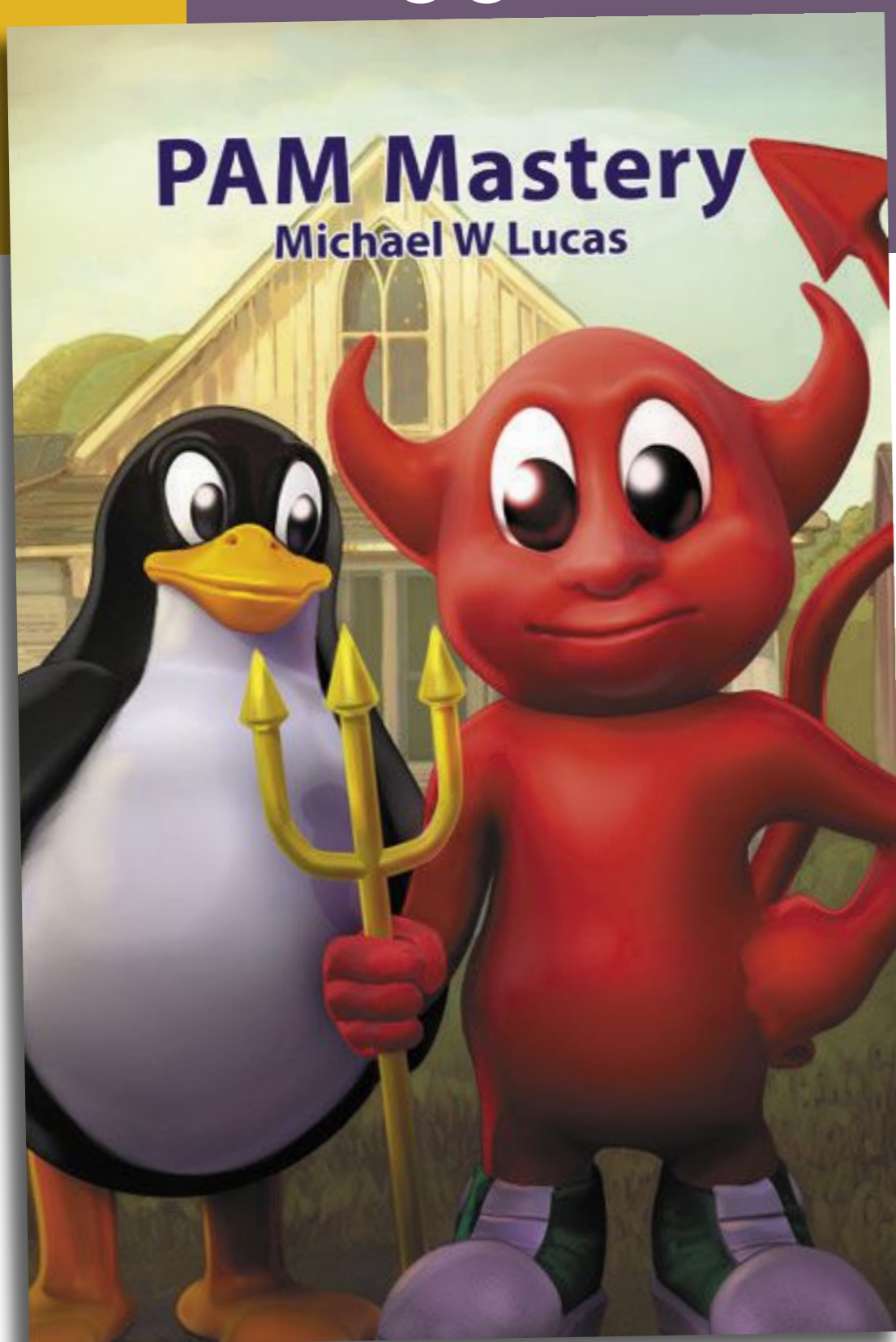
With our luck, though, we'll get one involving spreadsheets and Perl scripts.

Have a question for Michael?
Send it to letters@freebsdjournal.org



MICHAEL W LUCAS is the author of *Networking for System Administrators*, *\$ git commit murder*, and many others. His new books include *OpenBSD Mastery: Filesystems* and *Prohibition Orcs*. Get the entire interminable list from his SNMP OID or at <https://mwl.io>.

Pluggable Authentication Modules: Threat or Menace?



PAM is one of the most misunderstood parts of systems administration. Many sysadmins live with authentication problems rather than risk making them worse. PAM's very nature makes it unlike any other Unix access control system.

If you have PAM misery or PAM mysteries, you need PAM Mastery!

"Once again Michael W Lucas nailed it." — nixCraft

***PAM Mastery* by Michael W Lucas**

<https://mwl.io>



BOOK REVIEW

Understanding Software Dynamics by Richard L. Sites

REVIEW BY TOM JONES

Performance analysis is a difficult subject — a field where any claim must be backed up with a serious amount of rigor, detail of the work done, and where you must be very confident in your results.

In computer science we are almost dissuaded from worrying about performance issues. The frequently taught Knuth quote, *Premature optimization is the root of all evil* is used to keep junior developers on track to not worry too much about the small things.

Most of the time, this advice is given with good intentions. When you are learning, it is better to finish projects — to progress — rather than getting lost in minute details. But this has had the opposite effect of making information on improving performance of real systems hard to come by.

The full Knuth quote,

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

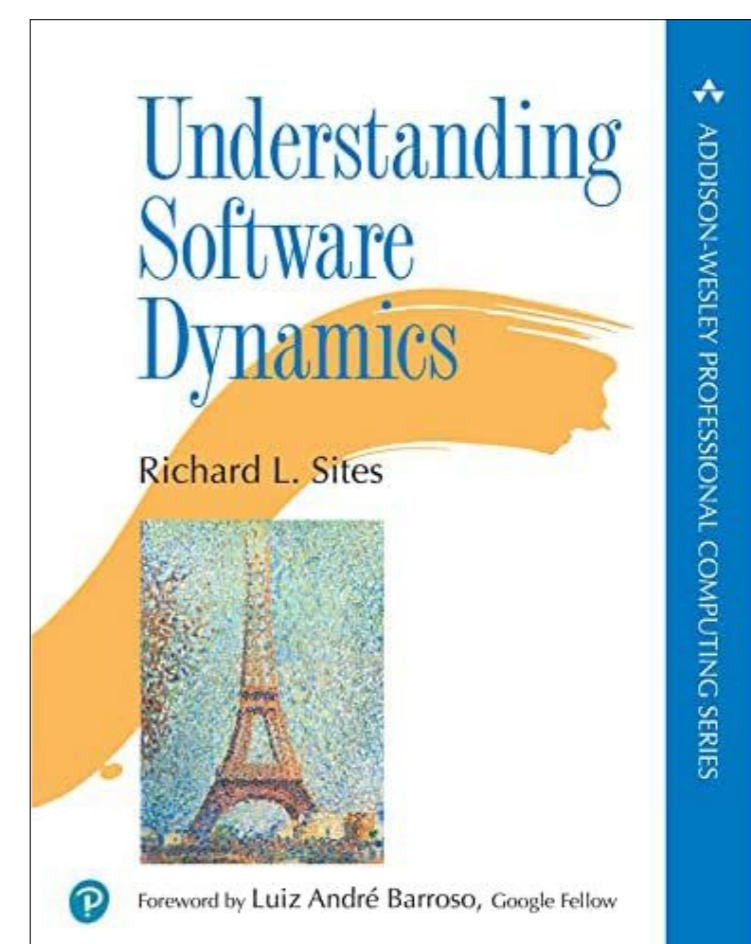
Yet, we should not pass up our opportunities in that critical 3%.

A good programmer will not be lulled into complacency by such reasoning, but he will be wise to look carefully at the critical code; but only after that code has been identified

tells us there are times where we should really care and our efforts spent on investigation will be rewarded.

There have been several performance books written in the last few years that I hold in high regard and have recommended many times. Brendan Gregg's most recent books *Systems Performance* (which acts to replace the long favored DTrace performance book) and *BPF Performance Tools* are well known and come from one of the world's leading experts in investigating and designing tools to better understand real-world, system performance. The lesser known, but excellent (and open source) *Performance Analysis and Tuning on Modern CPUs* by Denis Bakhvalov, dives deeper into why CPUs are slow and how computers can the best make use of them.

The Gregg books act as excellent introductory material for the performance expert to be. They offer a high-level view of looking at systems and using tools. *BFP Performance Tools* and *Systems Performance* offer methodology for the performance analyst and very



high-quality reference material that should be your first port of call when trying to debug an issue. With these two books and reading the source for the accompanying tools, you can learn how to build your own performance tools, but the books don't go into depth on what makes a good performance analysis tool.

Bakhvalov's *Performance Analysis and Tuning on Modern CPUs* shows how the perf set of tools can be used on Linux to explore programs. It introduces the core fundamentals a developer would need to know to understand the output of perf and how different factors influence software performance as analyzed with perf. The book is — again — a great introduction, but lacks solid examples (which are available as free course material from the author) and lacks the next steps for the developers who need their own custom tools to debug their problems.

Understanding Software Dynamics (USD) by Richard L. Sites stands out from the other recent works. It contains introductory material into how computers work and the factors that determine their performance, and it expands on this by practically describing the limitations that performance tools must meet to be useful to use on real systems.

The author is a veteran of the computer industry. He developed the first CPU performance instructions while at DEC and has had a long career investigating performance issues from CPU level bottlenecks up to entire data center wide application stacks while working with Google and Tesla.

USD acts as a practical introduction to exploring the performance of large systems as their dynamics change. In large systems, there are a lot of transactions moving through at a time — the performance of individual transactions is normally inconsequential. *USD* looks at the distribution of these transactions. Problems live with the outliers: Total systems performance is made of both the fast and the slow transactions and *USD* offers guidance to discover what the best- and worst-case performance should be in a system and how to explain what is happening when the 90th percentile is outside these bounds.

USD follows a data center RPC system as its core example and builds tooling to discover where performance issues can appear. The first 7 chapters look at measurement and how different system components contribute to final results. Here *USD* digs into very fine detail, offering guidance to understand why CPU instructions, memory, disk or network access might be the cause of performance bottlenecks in a system.

Throughout these explorations, the lesson is that looking at individual subsystems is not a good model for thinking about the performance of the entire system. Sites shows his thinking for establishing what the best- and worst-case performance of any computer subsystem should be and demonstrates how to test these estimates in the real world.

The second part of *USD* covers how to observe and measure the performance of real systems while maintaining acceptable levels of overhead. In this part, the reader learns different ways that the tools we use today are implemented and the opportunities for observation they provide. Here we are introduced to the design criteria for observation tools.

The third part builds on the lessons of the first two introductory parts of the book and

*Understanding
Software
Dynamics (USD)*
by Richard L. Sites
stands out from
the other recent
works.

shows a real implementation of the tracing ideas by introducing the KUTrace framework kernel — userspace tracing framework.

KUTrace is an example of a high bandwidth logging framework and has to be implemented with low overhead. It offers an extra source of information for debugging overly long running transactions.

KUTrace is a patch set for Linux that adds a framework for adding small ~64byte log entries to points in the kernel and userspace. The tracepoints are `__predict_false` branches by default. Once the KUTrace kernel module is loaded, they become active and start being saved into a buffer in the kernel.

USD goes into detail explaining the design of the log entries that KUTrace uses, the system interface and the operation of the kernel module.

Sites rounds out the text with 9 final chapters, each of which practically walks through a problem in one of the performance domains that book has examined. There is an example problem for a case of too much CPU execution, executing slowly and waiting for the CPU, Memory, Disk, Network, Locks, Time, and queues. Each of these chapters is a lesson acting on information presented earlier in the book.

Understanding Software Dynamics is an excellent addition to the library for anyone interested in a practical understanding of performance issues in real systems. It does a great job of introducing required background understanding without going into such depth that it is difficult to follow. Several chapters come with examples that help reinforce and expand on the lessons provided in the preceding text and they are focused on improving understanding.

Sites is clear in his explanation throughout, there are high quality figures taken from real systems at Google and from the example systems. Added color comes from the author's own experiences working on performance.

USD is a well written and carefully constructed book. It supports itself well and is approachable for a reader who hasn't previously delved into performance analysis. For a reader with a lot of experience, there is still much information to learn from here about the specifics of performance of different computer components and the new tooling the author introduces.

It does a great job of introducing required background understanding without going into such depth that it is difficult to follow.

TOM JONES, FreeBSD Developer and co-host of the BSDNow Podcast, wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting.



Keeping FreeBSD Secure: Learn the Whys and Hows with the FreeBSD Sec Team

BY PAM BAKER AND ANNE DICKISON

We all know the scene. The room is dark, with the only light provided by the laptop screen. The hooded figure is typing furiously at the keyboard. Suddenly, lines of symbols, letters, and numbers fly into the terminal window as the nefarious character smiles brightly. They are in.

But not so fast! The dogged security team has been planning for this. Protocols are in place. The breach is secured; the sinister hacker is captured; and of course, the world is saved. Ah, MovieOS. Don't you just love it?

Now we all know in the real world, trying to keep any type of technology secure is nearly a herculean task. Strengthening security for the FreeBSD Operating System is no different. But we wanted to know more about exactly what the FreeBSD Security Team does and why they do it. So, we sat down with Gordon Tetlow, a volunteer FreeBSD Security Officer, and Ed Maste, Deputy Security Officer, and Mark Johnston, a FreeBSD security team member. The latter two are sponsored by FreeBSD Foundation and support the security team in both ongoing operational aspects of the team's work, and proactive development.

Q: What is FreeBSD Project's overall approach to security?



Ed Maste: The security team focuses on several different aspects of security within FreeBSD. One area is what's often called a PSIRT, or Product Security Incident Response Team. This is a main focus of the security team today.

This team fields reports about vulnerabilities and issues and responds by identifying the problem and managing the release of the fix. Examples may include errors in drivers or pro-

protocols, bugs found by our own proactive fuzzing efforts, other automated tools, and code review. The team's response includes preparing or integrating patches to fix those issues, preparing and publishing security advisories to notify the community of the issue, and deploying binary updates.

A second focus area is proactive security work, which includes targeted efforts to find issues, vulnerability mitigations that reduce the impact when issues do occur, and general architectural security review. These efforts were historically undertaken directly by the security team. In the current security team model responsibility for certain areas has moved to separate groups of subject-matter experts. FreeBSD's random number generation subsystem is an example of one such area — the security team remains involved, but specific responsibility is delegated.

Proactive security work also includes ongoing code review and auditing, following security reports and discussions in other projects, fuzzing and test failure analysis, and related areas.

And a third area is the security of the FreeBSD infrastructure itself, meaning the FreeBSD website and source code repository and all the services that we run. In these cases, there are other groups within the project who have primary responsibility, while the security team may offer advice and review.

Proactive security work also includes ongoing code review and auditing.

Gordon Tetlow: The other role that we play is in coordination with industry efforts. There are vulnerabilities that affect more than just the FreeBSD project, where there is common code shared with other open source projects. We end up with industry-wide efforts to address those. An example would be OpenSSL, which is another project that we incorporate. We have to coordinate disclosure and coordinate patch response for that.

And then we do much the same all the way upstream, too. One example is when Intel had the "Spectre" and "Meltdown" speculative execution issues a couple of years ago. Literally everybody, every operating system manufacturer, and a lot of other folks all had to get together and coordinate an industry-wide response, for better or for worse. We play an important role in that broad industry response.

Q: What is FreeBSD's specific role in disclosures?

Ed Maste: If we have a vulnerability reported in FreeBSD that we will be addressing, we handle the public disclosure to the FreeBSD community of that vulnerability and handle the patch and binary update for the fix.

We also are involved in public disclosure in terms of coordination with industry partners and peers. If there's an issue that affects Linux and OpenBSD and NetBSD and FreeBSD, for example, and someone is coordinating the reporting of that issue with all of the different communities, then we'll collaborate with those other projects to help make sure that the fixes are released on the schedule set by the vulnerability reporter, or the industry peers who are managing and coordinating the issue.

Q: Do you have formalized roles, or a mission statement or charter guiding your security work?

Ed Maste: Quoting from the FreeBSD project's "[Administration and Management](#)" page,

The FreeBSD Security Team (headed by the Security Officer) is responsible for keeping the community aware of bugs, exploits and security risks affecting the FreeBSD src and ports trees, and to promote and distribute information needed to safely run FreeBSD systems. Furthermore, it is responsible for resolving software bugs affecting the security of FreeBSD and issuing security advisories. The [FreeBSD Security Officer Charter](#) describes the duties and responsibilities of the Security Officer in greater detail.

Gordon Tetlow: The security officer has an open-ended charter to make things secure, which includes the ability to override actions and decisions of other developers, if necessary, in the name of security. Now that's not something that we exercise lightly and it's definitely something we have to be very conscientious about using. But the charter mandates that we ensure, by whatever means necessary, that what we're doing is the right thing.

Q: How are reports on security advisories handled? Can they be anonymous and protected?

Ed Maste: We provide guidance on the FreeBSD [website](#) that describes the policies, the order of the approaches that the sec team follows, security advisories, and other helpful information.

Gordon Tetlow: Note the section "When is the security advisory considered" right [on the front page](#) for the security team. For people who are interested in reporting security advisories, there's also documentation on how to report a security advisory. That's listed kind of as a subset to that.

People can send us regular or PGP-encrypted email to secteam@FreeBSD.org. What we want to let people know is if they're wanting to get in touch with us on a sensitive issue, they're welcome to encrypt the data to us. That way they can know that only a certain couple of individuals would be able to read it.

The security officer has an open-ended charter to make things secure.

Q: How else do you find security issues?

Mark Johnston: We aim to find security problems proactively, in addition to addressing vulnerabilities reported by third-party researchers. We try to be proactive and responsive to all situations that arise.

In my case, it's largely just about day to day working on FreeBSD, looking at bug reports and user submissions and also doing my own testing. We have several developers in the community who spend most of their time doing nothing but testing FreeBSD and reporting bugs. Upon further examination like this, you might find a security vulnerability lurking in there, even if the person reporting it isn't aware of the security implications.

I spend a lot of time drilling down into those kinds of reports and looking for something that might be more serious than it appears at first glance.

Ed Maste: Mark also worked to bring the Syzkaller code-coverage-guided system call fuzzing tool to FreeBSD, and worked with the project's maintainers to have it run on a consistent basis. Syzkaller performs automated kernel fuzzing in order to find inputs that lead to a kernel crash or some inconsistency detected by instrumentation. Syzkaller's reports may indicate potential vulnerabilities, but in any case represent bugs to be fixed. Mark worked on increasing Syzkaller's code coverage, triaged its reports, and has fixed many issues as a result.

FreeBSD also has a stress testing suite, called "stress2", which can find race conditions or misbehaviour that occurs under high load. A number of kernel bugs have been fixed as a result.

Ed Maste: Among the useful cases that Mark identifies out of the general bug reports mailing lists, social media and other channels are issues that people have reported that they want fixed. Quite often they're unaware of the potential security impact of that problem. We try to understand and extend evaluation of the problem to include any potential security impact and act upon it as warranted.

Q: What's next for the security team?

Ed Maste: There are both technical and operational improvements we're looking at within the security team. We currently have focused efforts to discover potential issues via fuzzing and other tools. We intend to continue and increase this effort, for example by extending Syzkaller to include additional system calls, increasing code coverage.

This has been ongoing for some time, but we expect to increase our effort in revisiting system defaults, and applying sandboxing, privilege reduction, and other user space techniques to software in the base system and the ports collection.

Operationally, we are looking at improving coordination with downstream projects and vendors who use FreeBSD as the basis for their own development. We also need to keep working on bringing new members into the security team; this is a challenge shared by many open source projects.

The security officer has an open-ended charter to make things secure.

A prolific and versatile writer, **PAM BAKER** writes on many topics for leading tech and science publications. She is also the author of many dead tree books, ebooks and white papers. Her latest book is Decision Intelligence For Dummies which is about a new way to mine data and use AI in decision making. It was released in February 2022. Baker lives in Atlanta, Georgia where she's currently working on her first sci-fi novel.

ANNE DICKISON joined the Foundation in 2015 and brings over 20 years experience in technology-focused marketing and communications. Specifically, her work as the Marketing Director and then Co-Executive Director of the USENIX Association helped instill her commitment to spreading the word about the importance of free and open source technologies.



May Contain Hackers 2022

(MCH2022) A TRIP REPORT BY RENÉ LADAN

MCH2022 is a nonprofit, outdoor, hacker camp that took place in Zeewolde, the Netherlands, July 22 to 26, 2022. The event is organized by and for volunteers from the worldwide hacker community. Knowledge sharing, technological advancement, experimentation, connecting with hacker peers, and hacking are some of the core values of this event. MCH2022 is the successor of a string of similar events that have taken place every four years since 1989. These are GHP, HEU, HIP, HAL, WTH, HAR, OHM and SHA. <https://mch2022.org/>

DAY 0 • 7-21-2022 Thursday

Today is mostly about packing things and traveling up to Zeewolde where MCH 2022 is about to begin. I arrived in the afternoon and after checking in, I picked up my cardboard tent which will be my new home for the coming week.

I met most of the village "Frubar" which I will be part of. I got my badge working, but after some updates (yes, Wi-Fi was up in the late evening), the Python apps don't want to run. Dinner was a pizza salami which I bought at the food corner. After some drinks, it was time to go to bed.



DAY 1 • 7-22-2022 Friday

A coffee (thanks Frubar!) and a shower made me feel human again, after which I started typing the first bits of this trip report. Today, the uplink for the Ethernet arrived, providing almost gigabit speeds.

I visited some of our "remote" village members who had their camper with them and consequently were in the camper area. Today was the official first day, but I missed the opening talk because we were busy setting up the UbaBot (admittedly, I mostly watched, as it took quite some preparation to get it up and running).

This "bot" is an automated cocktail making machine, from which you can select and enjoy various pre-programmed cocktails, given that you feed it the right ingredients, of course. You can find more about this machine on the Internet.

Evening shows are planned after the talks and workshops, but today I stayed in the village tent and played a few rounds of Uno with some fellow villagers.





DAY 2 • 7-23-2022 Saturday

Today I visited some talks and workshops: a talk by Karsten Nohl on hacking 5G networks with OpenRAN, a KiCad workshop about designing (but not actually making) PCBs, and a workshop on programming the camp badge.

Things never go as planned, Karsten's talk was somewhat during lunch time and therefore overruled. At Frubar, lunch did not consist of a quick sandwich. Some people in the village were quite enthusiastic about the grill we brought, so we decided to have grilled steak for lunch this week.

Regarding the KiCad workshop, I found myself trying to set up the Espressif IDE in a Ubuntu chroot under FreeBSD and getting communication with the badge up and running. The badge comes with some Python script for file manipulation, which did work in the end.

I arrived at the badge workshop in the DNA tent, which was already filled to the brim, but I managed to find a seat and the fellow sitting next to me and I experimented with the badges together.

The workshop was targeting the embedded Python that the badge runs. Their tutorial included a program to draw random lines which turned out to run fine.

After the workshop, I wandered back to base. Later in the evening, I stumbled upon a performance of the Ambrassband that the organization had scheduled. Live brass music on-stage.



DAY 3 • 7-24-2022 Sunday

I didn't plan many talks or workshops for today, however, today was also Tor day. I attended the Tor talk by Alexander Færøy and the unofficial Tor relay operator meetup in the evening.

After lunch, I went to the exhibition of the Home Computer Museum in the retro tent and finally played Duckhunt after all these years. I also had a chat with a guy from the museum about, well, old computers ;). Later that evening, some of us went to visit the party at the silent disco, after saying hello to the folks at the Geraffel tent, who also had a small party.

DAY 4 • 7-25-2022 Monday

I attended some talks today: one about Tesla cars and the security of its keys/phone app by Martin Herfurt, one about reporting vulnerabilities by the Dutch Institute for Vulnerability Disclosure (we watched this on the big screen in the village tent), a talk about separate audio without physical walls by Adrian Lara Moreno (also in the tent — the demo didn't work out I guess), and drscream's talk about Illumos zones.

I also went to the Area 42 Workshops, which was mostly just a small tent set up as a classroom to watch the talk about making a drone out of the camp badge (or how it failed).

I stayed at the village tent this evening (sorry Symphony of Fire show) and played some more Uno.



DAY 5 • 7-26-2022 Tuesday

Today is the last day of this event and apparently the time when all camps start to pack up after breakfast. This is also the day that my temporary home will be recycled by the company that sells them to MCH, so I'd better pack up too.

I had some events scheduled, but those were overruled because of saying goodbye to those in the Frubar group who were also heading home.

One last thing: I need to have the sound selection switch of my badge replaced by one with a knob to grab, as the old knob mysteriously broke off, perhaps during the workshop on Saturday.



Back Home • 7-27-2022 Wednesday

I woke up in my apartment and realized there was no more steak for lunch :(

Back home, I restored the regular 'rene' user on my laptop and registered the badge on my home Wi-Fi.

Some firmware and application updates came in (everything works again ;)), it now runs OS version 1.4. Under FreeBSD, it shows up as:

```
ugen0.5: <Badge.team MCH2022 badge> at usb0
umodem0 on uhub0
umodem0: <ESP32 console> on usb0
umodem0: data interface 1, has no CM over data, has no break
umodem1 on uhub0
umodem1: <FPGA console> on usb0
umodem1: data interface 3, has no CM over data, has no break
```

The RP2040, which connects the ESP32 to the USB port and knobs, is unlisted.

The next event will be held in 2025, but next year a sibling event will be held in Germany.

Almost all videos are available at <https://media.ccc.de/b/conferences/camp-NL/mch2022/>

RENÉ LADAN studied computing science at the Eindhoven University of Technology where he graduated in 2006. He has worked at various companies, including the university itself. He currently works as a software engineer at Carapax IT.

When not doing BSD stuff and still in nerd mode, he likes to tinker with his homebrew time station receivers. Outside of technical things, René likes to hike, puzzle, and work in his parents' garden.



Events Calendar

BSD Events taking place through February 2023

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.



November 2022 FreeBSD Vendor Summit

November 3-4, 2022

Virtual

FreeBSD

<https://wiki.freebsd.org/DevSummit/202211>

Join us for the online November 2022 FreeBSD Vendor Summit. The event will consist of virtual, half day sessions. In addition to vendor talks, we will have discussion sessions and a separate hallway track. The vendor summit is sponsored by the FreeBSD Foundation.



FOSDEM 2023

February 4-5, 2023

Brussels, Belgium

<https://fosdem.org/2023/>

FOSDEM is a two-day event organized by volunteers to promote the widespread use of free and open source software. The event offers open source and free software developers a place to meet, share ideas and collaborate. Renowned for being highly developer-oriented, FOSDEM brings together some 8000+ developers from all over the world.

FreeBSD Office Hours

<https://wiki.freebsd.org/OfficeHours>

Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions.

Past episodes can be found at the FreeBSD YouTube Channel.

<https://www.youtube.com/c/FreeBSDProject>.