MAGAZINE

# BSD

FOR NOVICE AND ADVANCED USERS

# TUNING ZFS ON FREEBSD

INSIDE

POSTGRESQL PARTITIONING
SECURING DNS TRANSACTIONS
MPD5 – VPN SERVER WITH FREEBSD SETUP AND MANAGEMENT
MAHESHABSD SERVER EDITION HAS BEEN JUST RELEASED!

VOL.6 NO.08
ISSUE 08/2012(37)
1898-9144

FreeNAS 8 POWERED

# TrueNAS™

## UNIFIED. SCALABLE. FLEXIBLE.


Across all industries the demands of data infrastructure have soared to new heights.

As capacity requirements continue to rise at an ever-increasing rate, performance must not be compromised. The hybrid architecture and advanced software capabilities of the TrueNAS appliance enable users to be more agile, effectively manage the explosion of unstructured data and deploy a centralized information storage infrastructure. Whether it's backing virtual machines, business applications, or web services, there's a TrueNAS appliance suited to the task.

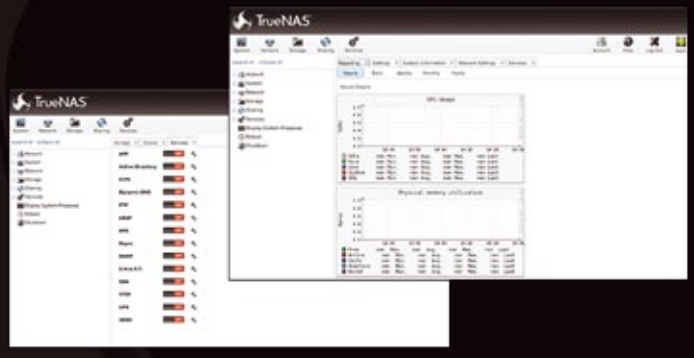
# TrueNAS™ Storage Appliances: Harness The Cloud

## iXsystems' TrueNAS Appliances offer scalable high-throughput, low latency storage

All TrueNAS Storage Appliances feature the Intel® Xeon® Processors 5600 series, powering the fastest data transfer speeds and lowest latency possible. TrueNAS appliances come in three lines: Performance, Archiver, & High Availability. High-performance, high-capacity ioMemory modules from Fusion-io are available in the TrueNAS Enterprise, Ultimate, and Archiver Pro models.

## Key Features:

- **One or Two Six-Core Intel® Xeon® Processors 5600 series**
- **Share Data over CIFS, NFS and iSCSI**
- **Hybrid storage pool increases performance and decreases energy footprint**
- **128-bit ZFS file system with up to triple parity software RAID**

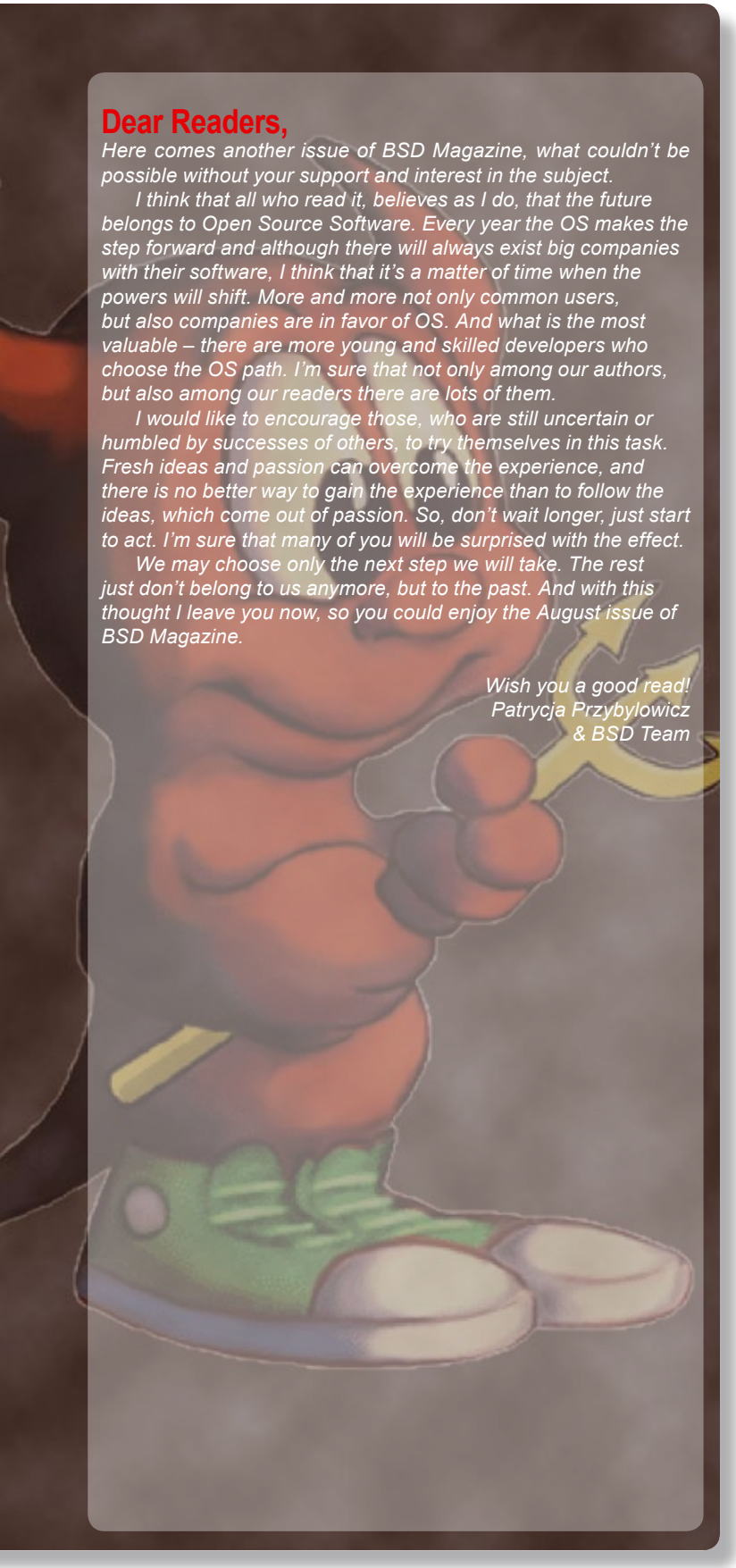



*Optional component

| FEATURES | TrueNAS Pro | TrueNAS Enterprise | TrueNAS Ultimate | TrueNAS Fileshare | TrueNAS Archiver Pro | TrueNAS Pro-HA | TrueNAS Enterprise-HA | TrueNAS Ultimate-HA |
|---|---|---|---|---|---|---|---|---|
| | PERFORMANCE | | | ARCHIVER | | HIGH AVAILABILITY | | |
| Fusion-io Card | | X | X | | X | | | |
| Deduplication | | | | | X | | | |
| High Availability | | | | | | X | X | X |
| Gigabit NICs | Quad | Dual | Dual | Dual | Dual | Quad | Quad | Dual |
| 10 Gigabit NICs | | Dual* | Quad* | | Dual* | | | Quad* |
| Max Main Memory | 48Gb | 96Gb | 192Gb | 48Gb | 192Gb | 48Gb | 96Gb | 192Gb |
| Max Capacity | 320TB | 500TB | 450TB | 680TB | 2.2PB | 250TB | 310TB | 1.4PB |
| Rack Units | 2U/4U | 2U/4U | 4U | 2U/4U | 4U | 3U | 3U | Dual 3U |

**Call iXsystems toll free or visit our website today!**

**1-855-GREP-4-IX | www.iXsystems.com**



## Dear Readers,

*Here comes another issue of BSD Magazine, what couldn't be possible without your support and interest in the subject.*

*I think that all who read it, believes as I do, that the future belongs to Open Source Software. Every year the OS makes the step forward and although there will always exist big companies with their software, I think that it's a matter of time when the powers will shift. More and more not only common users, but also companies are in favor of OS. And what is the most valuable – there are more young and skilled developers who choose the OS path. I'm sure that not only among our authors, but also among our readers there are lots of them.*

*I would like to encourage those, who are still uncertain or humbled by successes of others, to try themselves in this task. Fresh ideas and passion can overcome the experience, and there is no better way to gain the experience than to follow the ideas, which come out of passion. So, don't wait longer, just start to act. I'm sure that many of you will be surprised with the effect.*

*We may choose only the next step we will take. The rest just don't belong to us anymore, but to the past. And with this thought I leave you now, so you could enjoy the August issue of BSD Magazine.*

*Wish you a good read!*
*Patrycja Przybylowicz*
*& BSD Team*

MAGAZINE BSD

**Editor in Chief:**
Ewa Dudzic
*ewa.dudzic@software.com.pl*

**Supportive Editor**
Patrycja Przybyłowicz
*patrycja.przybylowicz@software.com.pl*

**Contributing:**
Paul Ammann, Juraj Sipos, Martin Matuska, Luca Ferrari, Antonio Francisco Gentile

**Top Betatesters & Proofreaders:**
Paul McMath, Bjørn Michelsen, Barry Grumbine, Babak Farrokhi, Eric De La Cruz Lugo, Eric Geissinger , Luca Ferrari, Imad Soltani, Norman Golish, Luiz Claudio Pacheco, Radjis Mahangoe, Rob Cabrera, Will Clayton, Zander Hill

**Special Thanks:**
Denise Ebery

**Art Director:**
Ireneusz Pogroszewski

**DTP:**
Ireneusz Pogroszewski
*ireneusz.pogroszewski@software.com.pl*

**Senior Consultant/Publisher:**
Paweł Marciniak *pawel@software.com.pl*

**CEO:**
Ewa Dudzic
*ewa.dudzic@software.com.pl*

**Production Director:**
Andrzej Kuca
andrzej.kuca@software.com.pl

**Executive Ad Consultant:**
Ewa Dudzic
*ewa.dudzic@software.com.pl*

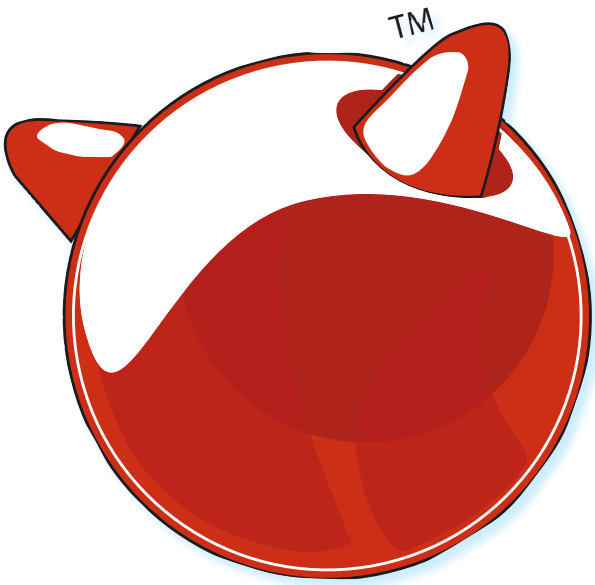
**Advertising Sales:**
Patrycja Przybyłowicz
*patrycja.przybylowicz@software.com.pl*

**Publisher :**
**Software Press Sp. z o.o. SK**
**ul. Bokserska 1, 02-682 Warszawa**
**Poland**
worldwide publishing
tel: 1 917 338 36 31
*www.bsdmag.org*

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

## What's New






## How To



## Security



# MaheshaBSD

## Server Edition Has Been Just Released

Many newcomers to FreeBSD find it difficult to setup their own FTP/WWW server quickly and, on the other hand, experienced users sometimes need to take precautions for unexpected crash situations – that is, to have a strategy for time economization and portability, as these two are valuable assets in our rushing world.

**What you will learn…**

- How to run a simple and smart FTP/WWW server.

**What you should know…**

- How to mount drives in FreeBSD.

A USB flash drive with an operating system running off it answers many questions when portability and time effectiveness become indispensable. All you need to do is to plug your USB flash drive into the USB port. The computer's operating system stays unaltered and geared up for any task it is used for. And USB flash drives boot several times faster than CD's or DVD's.

Readers who will only randomly find this article should know that MaheshaBSD and MaheshaBSD Server edition are two different things, although tightly related to each other. The article about MaheshaBSD was already published in the March 2012 issue of BSD Mag (*http://bsd-mag.org/magazine/1795-nessus-exploitation-tools-and-payloads*).

### The Story Behind MaheshaBSD Server

A year ago I bought a second-hand IBM notebook and the experienced technician in the store there told me, after we discussed which operating system to use, that he would call me if he had a customer interested in an FTP/WWW server running FreeBSD. I welcomed the offer. However, I was surprised to learn that it was difficult for a proficient Linux user to setup an FTP server on FreeBSD.

Some time later a friend asked me if I knew of any good FTP server software for Windows. Then another friend asked me to recommend him some software which counseling psychologists could use to share data. As many

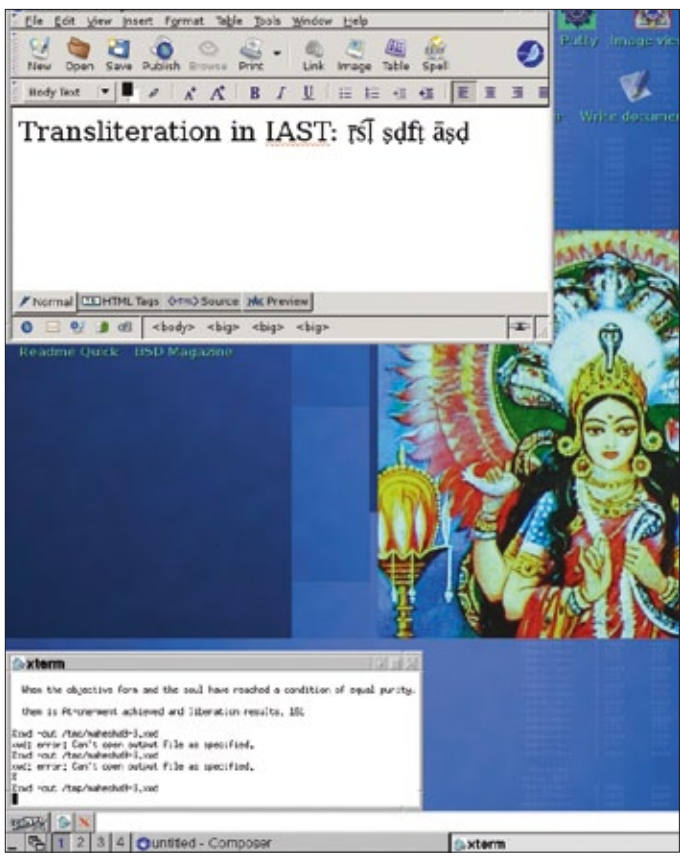



**Figure 1.** *In the server edition of MaheshaBSD transliteration of Sanskrit works too (Seamonkey)*

Windows users consider Unix to be the unknown ocean in which they do not want to dip their toes, my only choice was to look for Windows software. I looked on the Internet and was stunned by the prices people are willing to pay for Windows FTP server solutions. Some of them climb up even up to 500 US dollars. My head whirled round. Thus, the idea to make my own easy FTP/WWW server solution was born and finally MaheshaBSD Server edition slithered to day light – a smart and straightforward FTP/WWW server, something I was looking for. It runs off a USB flash drive and it does not require a Windows license. It is free for personal use.

A decision to sell my FTP/WWW server solution has two goals:

- A wish to donate some money to FreeBSD and OpenBSD (sorry I do not mention other BSD systems here; I am only familiar with the two mentioned above).
- I have a few friends in very poor countries and I would like to help them (Figure 1).

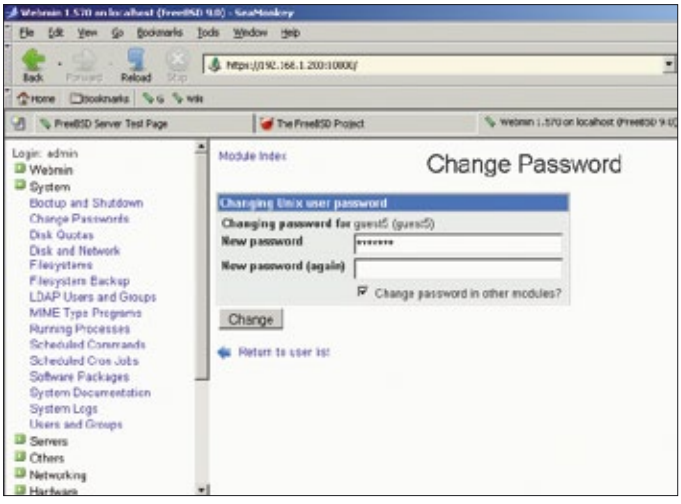



**Figure 2.** *In MaheshaBSD Server even a child can change passwords in Webmin*

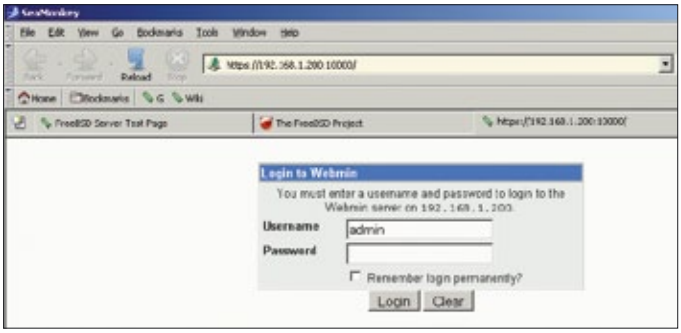



**Figure 3.** *To log in to Webmin, use any of the addresses specified above, as you see on the picture*

## Unique Features of MaheshaBSD Server

All users, whether experienced or not, whether friends of Unix or not, will have an immediate possibility to operate a simple FTP/WWW server off a USB flash/hard drive. Portability and time effectiveness are vital assets also for experienced users. If you work in a small business and your hard drive or computer with a server system on it fails to boot one day (imagine a situation your boss does not want to invest money in an expensive IT infrastructure), it may take several hours to buy a new hard drive, install your favorite server system on it, and configure it.

USB flash drives are very fast and with MaheshaBSD Server you will get the operating system (FreeBSD) and a superbly easy FTP/WWW server into which you just copy files via SFTP. A prospect to use NTFS (and FAT32) disks/flash drives with write access and use them as a place for immediate FTP/WWW data storage will appear very practical not just for Windows users. Please do not ridicule me for the above statement, as for many people out there it is still much easier to use a spare NTFS/FAT32 drive in Unix (without much knowledge about Unix) than to format it, as the data on it may have a value and backing it up takes time too. Unfortunately, not many BSD LiveCD/USB projects support writing to NTFS drives.

### Unique features

MaheshaBSD Server supports quotas.

MaheshaBSD Server has a remote administration tool (Webmin; Figure 2).

You can work with any hard disk (NTFS partitions too).

MaheshaBSD Server (FTP/WWW/Webmin) is connectable on aliased IP's on LAN that all end with 200 and

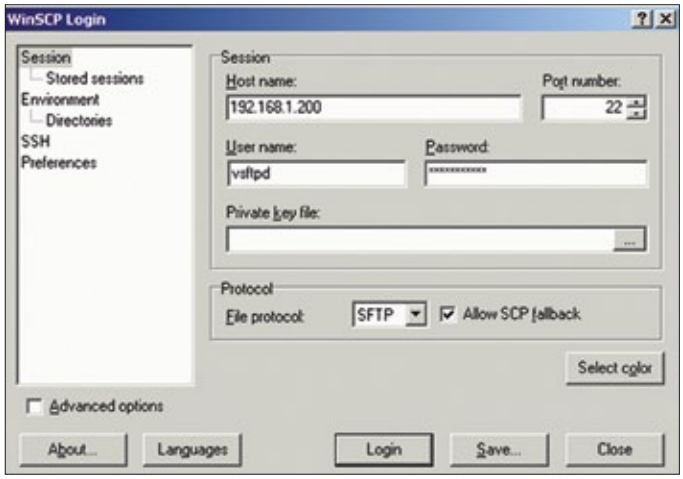



**Figure 4.** *To fetch your passwords, you must log in as user guest5 and not as user vsftpd, as you see on the picture*

which all can be used as follows (depending on your LAN configuration): 10.0.0.200, 10.0.1.200, 192.168.0.200, 192.168.1.200, 172.16.0.200, and 172.16.1.200 (Figure 3).

SSH works with the "boss" account only. This is a security measure.

## A Practical Example How To Use The MaheshaBSD Server's FTP/WWW Server

The text is written also with focus on absolute beginners.

Download the USB image from the following URL: *ftp://2227.x.rootbsd.net/index.html*.

Unrar it and write it to your USB flash drive or USB hard drive.

In FreeBSD:

```
dd if=/path/MaheshaBSD9-server.img of=/dev/da0
                    bs=10240 conv=sync
```

In Windows use a program such as Winimage.

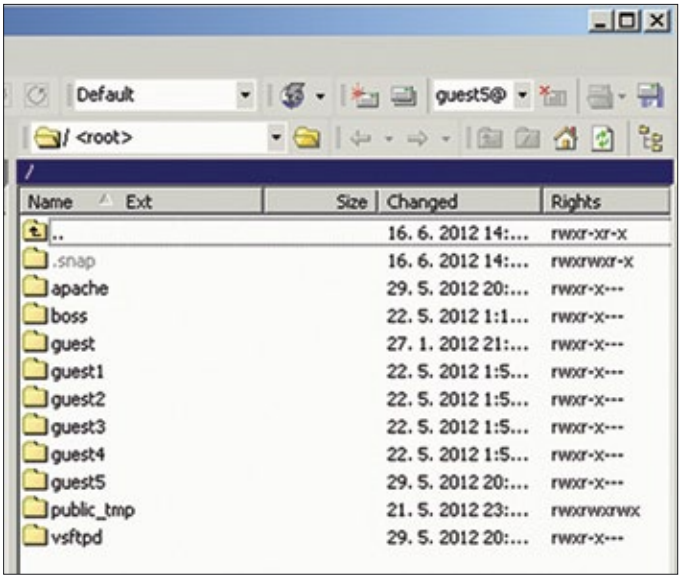



**Figure 5.** *After you log in, you will see all the user directories in /home with one shareable directory (public_tmp)*

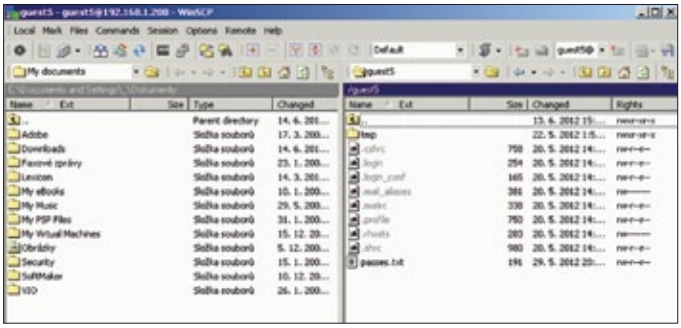


**Figure 6.** *To see your passwords, first click on the guest5 directory and then on passes.txt*

When you boot your computer with MaheshaBSD Server, simply use programs such as Winscp for Windows (only SFTP protocol works), or SFTP in Unix, and fetch your passwords (log in as guest5):

- Login: guest5
- Password: guest6

Then just log in to your vsftpd account (via SFTP) and copy anything to the vsftpd > ftp directory (Figure 4-6).

Of course, you can do all of the above steps also with physical access to the computer where MaheshaBSD Server is running. When you boot your computer off your USB flash/hard drive, you will see the password for root in a blue text. Then just log in and type (Figure 7 and Figure 8):

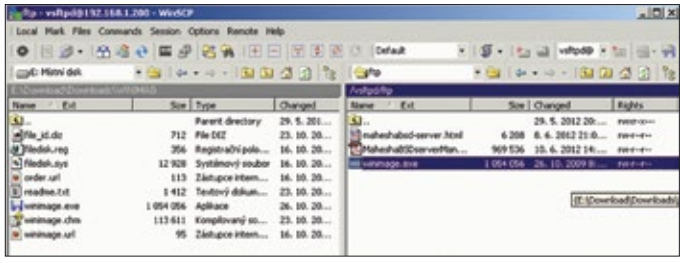


**Figure 7.** *All you need is to copy any file to the vsftpd > ftp (or apache > www) directory to have it immediately displayed in your browser*


**Figure 8.** *We chose the file winimage.exe as an example here; the file is immediately displayed in the MaheshaBSD Server's FTP server*


**Figure 9.** *Even little children will create html pages in easy to use Seamonkey, which is available both in MaheshaBSD and MaheshaBSD Server*

I also tested this with a NTFS drive redirected via mount_nullfs and I copied a 700 MB file onto it without any problems.

All other details, if you are more interested in this project, are in the documentation available at: *http://www.freebsd.nfo.sk/MaheshaBSDserverManual.pdf*.

## Conclusion

Groups tend to live inside themselves and often without knowledge of other waters. Thus, the MaheshaBSD Server's goal is not to be penchant for one group only. To attract new users to the BSD world requires giving them some of the water they were accustomed to bathe in. If a child sets up IP Forwarding in its router and will easily start its own WWW/FTP server from home, this may be its first challenge to look at FreeBSD. To easily share data can be the same challenge for scholars, teachers, small businesses, and actually for all Windows users, too, if they happen to ask questions where to find a simple FTP/WWW server solution. Not many non-Windows choices (although usable with Windows) await them on their market with FTP/WWW servers. I made this work also with hope that skilled Linux users (like the technician mentioned in the beginning of this article), who do not know how to set up an FTP/WWW server with FreeBSD, might be also challenged to look at this project.

I thank *http://www.rootbsd.net* for allowing me to distribute MaheshaBSD and MaheshaBSD Server.

**JURAJ SIPOS**

***Juraj lives in Slovakia and he works in a library in an educational institute. Some time in the past he was fortunate to travel around the world and he spent a bit of time in India and Australia. Juraj's hobbies are computers, mostly Unix, but spirituality too. His first published computer article was Xmodmap Howto (http://tldp.org/HOWTO/Intkeyb/). In addition to computers, he is very interested in Hinduism but not really the guru side of things, but more-so freedom and self-actualization. More at his website:***

***http://www.freebsd.nfo.sk/ (FreeBSD)***

***http://www.freebsd.nfo.sk/maheshaeng.htm (MaheshaBSD)***

# Tuning ZFS on FreeBSD

ZFS is a modern 128-bit file system based on the copy-on-write model. It originates from the OpenSolaris project and has first appeared in FreeBSD in 2008. ZFS has many innovative features including an integrated volume manager with mirroring and RAID capabilities, data checksumming and compression, writable snapshots that can be transferred between systems and many more.

**What you will learn…**

- how to optimize ZFS for various applications and workloads
- how to measure and evaluate ZFS cache efficiency

**What you should know…**

- ZFS system administration basics
- working with sysctl(8) and loader(8) tunables

In this article I am going to discuss several tuning options including sysctl(2) knobs and give examples how can ZFS performance and efficiency can be measured and evaluated. This article is intended for FreeBSD users with ZFS version 28 available since 8.3-RELEASE and 9.0-RELEASE.

ZFS has a lot of tuning options accessible via the sysctl(8) command. In addition, the ZFS part of the OpenSolaris kstat (kernel statistics facility) framework has also been made available on FreeBSD providing raw statistical data in form of various counters. There are more than 60 vfs.zfs knobs and over 80 kstat.zfs knobs providing access to miscellaneous kernel counters, state and sizing variables. There is currently only a very limited amount of tools available that process this information.

The first sections of this article are going to focus on tuning zfs prefetch and caches, introducing the zfs-stats and zfs-mon statistics processing tools for measurement and evaluation. They are followed by individual tuning tips like using ZFS on web, database or file servers, and optimizations for the advanced format (4k sector) drives.

## General Tuning Tips

### RAM memory

The amount of system RAM has a significant impact on ZFS performance, especially if using deduplication. Many issues can be cured by increasing system's RAM memory. The recommended memory minimum is 1GB, but I suggest for highly utilized setups at least 8GB of system RAM. For server use error correcting ECC memory is an advantage.

### Access time

On a FreeBSD system, every time a file is accessed its access time (atime) gets updated. This may generate a lot of disk write activity on servers working with a large number of files. In such a case you might want to disable atime (access time) for the affected datasets or for the whole pool. Turning off atime may improve performance of on all types of application servers.

```
# zfs set atime=off dataset
```

### Dataset compression

Using ZFS dataset compression saves space but has a negative effect on system's CPU performance and responsiveness. On the other hand, enabling compression (mainly LZJB) may increase your data throughput, especially for slow storage. In particular the gzip compression costs essentially more CPU time then the less-compressing LZJB compression. Therefore I recommend using dataset compression only if you have compressible data and the dataset is not a performance bottleneck or if you are generally low on storage space. Datasets with low activity

containing e.g. log files are good candidates for gzip compression. If you have fast storage with enough space and need top-notch performance, disable dataset compression for the affected datasets.

```
# zfs set compression=[on|off] dataset
```

### Deduplication

ZFS deduplication is a relatively new feature that enables you to save space by keeping a single copy of data that is available on your ZFS dataset in multiple copies. Deduplication requires a large amount of RAM memory. The ideal situation is if your whole deduplication table fits into memory otherwise you may experience decreased system performance. Deduplication can be enabled and/or disabled on a per-dataset basis:

```
# zfs set dedup=on
```

You can view (detailed) deduplication information about a pool using the zdb command:

```
# zdb -D pool
```

or

```
# zdb -DD pool
```

It is possible to simulate the effect of enabling deduplication for a pool. To verify possible space gains, a resulting deduplication ratio of more than 2.00 indicates a good candidate for deduplication:

```
# zdb -S pool
```

A discussion of the benefits and costs of ZFS deduplication is available at the blog of Constantin Gonzales (*http://constantin.glez.de/blog/2011/07/zfs-dedupe-or-not-dedupe*).

### ZFS send/receive

If you are using send and receive in the same run (if you are not streaming into a file but piping zfs send directly to zfs receive) then you should consider using a buffering solution to speed up the process. I personally recommend the "mbuffer" program available in the FreeBSD ports tree as `misc/mbuffer`. Mbuffer allows you to buffer local streams (via a pipe) and is capable of standalone network streaming.

```
# zfs send tank/a@s1 | mbuffer -m 128M | zfs receive
                tank2/a@s1
```

## Cache and Prefetch Tuning

### Adaptive Replacement Cache (ARC)

One of the primary tunable ZFS features is the memory-based *Adaptive Replacement Cache* (ARC). Data and metadata of blocks read from disk devices are stored in this cache. ARC provides a major speedup to ZFS operations and is enabled by default.

The main loader(8) tunables for ARC are:

- `vfs.zfs.arc_max`: Maximum ARC size (in bytes)
- `vfs.zfs.arc_min`: Minimum ARC size (in bytes)
- `vfs.zfs.arc_meta_limit`: ARC metadata limit (in bytes)

With these tunables you can control the size limits of the ARC cache on your system. The minimum and maximum values for ARC are automatically sized on system boot, depending on the installed RAM memory. The default values are:

- `vfs.zfs.arc_max`: physical RAM less 1 GB (or `vm.kmem_size`, whatever is smaller)
- `vfs.zfs.arc_meta_limit`: 1/4th of `arc_max`
- `vfs.zfs.arc_min`: half of `arc_meta_limit` (equals to 1/8th of `arc_max`)

Both `arc_min` and `arc_max` have a hard-coded minimum of 16MB and both `arc_meta_limit` and `arc_min` may not be higher then `arc_max`.

To display your current ARC size (in bytes), run:

```
# sysctl kstat.zfs.misc.arcstats.size
```

Default values should be sufficient for most users, as if your system requires memory for other tasks, the ARC memory is automatically freed, shrinking down to the `arc_min` value. Changing `vfs.zfs.arc_max` is of advantage only if you need explicitly reserved memory for other use. Alternatively there may be situations where your metadata cache gets filled up (e.g. you have lots of small files) and you observe bad performance of your ARC. In such a situation you may want to increase the `arc_meta_limit` (loader(8) tunable). Please note that `arc_min` is enforced to be at least the half of `arc_meta_limit`.

To display your current ARC metadata usage and metadata limit (in bytes), run:

```
# sysctl vfs.zfs.arc_meta_used
# sysctl vfs.zfs.arc_meta_limit
```

By default, ARC is enabled for all datasets. If you experience performance problems because of full and

inefficient ARC you can decide to disable ARC or limit it to cache only metadata on non-critical datasets:

```
# zfs set primarycache=[all|metadata|none] dataset
```

For displaying uptime and real-time ARC activity and efficiency, please refer to the "zfs-stats and zfs-mon" section of this article.

## Level 2 Adaptive Replacement Cache (L2ARC)

Fast block devices (e.g. SSD drives) can be used to extend the ARC cache of specific ZFS pools. This cache is called *Level 2 Adaptive Replacement Cache* (L2ARC). This cache type is optimal mainly in write-once and read many storage scenarios, e.g. the static content of internet websites with image and video files that do not get overwritten (but the total space used keeps growing). It requires fast cache devices (use of fast SSD drives is recommended). On the contrary to ARC being shared for the whole system, L2ARC devices are pool-bound and cache only the data of the pool they are attached to.

The size of L2ARC cache is defined by the size of the cache device(s). To add/remove cache devices from a pool, you can use the following commands:

```
# zfs add pool cache device
# zfs remove pool device
```

L2ARC provides several system tunables, I am going to explain the following:

```
vfs.zfs.l2arc_feed_again: turbo warmup
vfs.zfs.l2arc_feed_secs: interval secs
vfs.zfs.l2arc_write_max: max write size
vfs.zfs.l2arc_write_boost: extra write during warmup
vfs.zfs.l2arc_headroom: number of dev writes to precache
vfs.zfs.l2arc_noprefetch: don't cache prefetch bufs
```

The first tunable above enables or disables L2ARC turbo warm-up. The turbo warm-up phase happens between system bootup and the L2ARC cache getting "warm" (= the first time L2ARC evicts data). During this phase the ARC write speed is calculated as `l2arc_write_max + l2arc_write_boost` in bytes every `l2arc_feed_secs` seconds. If the warm-up phase is disabled or L2ARC is already in warm state, data is written at a maximum speed of `l2arc_write_max` bytes every `l2arc_feed_secs` seconds. Default settings are warm-up enabled, `l2arc_feed_secs` set to one second and both `l2arc_write_max` and `l2arc_write_boost` set to 8MB. The `l2arc_headroom` tunable defines the number of L2ARC writes to be precached. The default value is 2.

The goal of these settings is to avoid overwriting SSD devices too quickly as these have a limited number of overwriting cycles. These settings have been defined in 2008 and modern SSD drives can easily operate at faster speeds. There are recommendations to at least double the `vfs.zfs.l2arc_write_max` and `vfs.zfs.l2arc_write_boost` loader(8) tunables to 16MB.

The `l2arc_noprefetch` tunable is set to 0 by default. This disables caching of sequential reads. Enabling this setting (value of 1) does in many cases improve the L2ARC performance, e.g. for video streaming or web-serving of large files.

```
# sysctl vfs.zfs.l2arc_noprefetch=1
```

By default, L2ARC is enabled for all datasets. You can disable L2ARC or limit it to cache only metadata as a per-dataset setting:

```
# zfs set secondarycache=[all|metadata|none] dataset
```

For displaying uptime and real-time L2ARC activity and efficiency, please refer to the "zfs-stats and zfs-mon" section of this article.

## ZFS Intent Log (ZIL)

The ZFS Intent Log provides synchronous semantics for ZFS. It is used to guarantee data consistency on fsync(2) calls. It is used to replay data transactions in case of a kernel panic, hardware or power failure.

By default, the ZFS Intent Log takes up a small amount of storage space of each ZFS pool. To speed up synchronous writes, a separate log device (including a mirrored device) may be used. Fast SSD drives are recommended for this purpose.

To add or remove a log device to a pool, use the following commands:

```
# zpool add tank log device
# zpool remove tank device
```

It is possible to configure the synchronicity behaviour on a per-dataset setting:

```
# zfs set sync=[standard|always|disabled] dataset
```

By setting sync to disabled, data is written to storage only on periodical (TXG – *Transaction Group*) write times. This improves write performance but introduces the risk of losing data during a kernel panic, hardware or power failure and makes this option interesting only for tempo-

rary and volatile data. Setting sync to always has a large performance penalty.

## File-level Prefetching (zfetch)

The file-level prefetching mechanism implemented in ZFS is named "zfetch". This mechanism analyses read patterns of files and tries to predict next reads resulting in reduction in application response times. In some workloads, zfetch may be CPU-intensive and limit scalability. The efficiency of prefetch can be displayed and monitored with the "zfs-stats" and "zfs-mon" tools (discussed later). Zfetch is enabled by default (disabled on systems with less than 4GB of RAM) and you may disable or re-enable it by setting the following loader tunable to 1 (disable) or 0 (enable):

```
vfs.zfs.prefetch_disable: Disable prefetch
```

## Device-level Prefetch (vdev prefetch)

The vdev prefetch mechanism does pre-read data after small reads from pool devices. Currently, the vdev prefetch cache is disabled by default. Long-term use has shown that it is inefficient in most cases and its memory consumption is proportional to the number of vdevs on a system. There are user reports that re-enabling vdev cache may significantly speed up the scrub (and resilver) process on raid-z devices (or systems with slow drives) by reducing disk seek times and speeding up metadata reads.

The vdev prefetch is primarily controlled by the vfs.zfs.vdev.cache.size loader(8) tunable that contains the per-vdev cache size in bytes and is disabled by default (value of 0). For expreienced users, there are additional tunables for fine-tuning available under the `vfs.zfs.vdev.cache` loader(8) tunable group.

```
# sysctl -d vfs.zfs.vdev.cache
```

To enable vdev prefetch, set `vfs.zfs.vdev.cache` in loader.conf(5) to a desired size in bytes different from zero, e.g. previous default value of 10 megabytes:

```
vfs.zfs.vdev.cache.size=10485760
```

**Listing 1.** *Example output of "zfs-stats"*

```
ARC Size:                               79.89%  25.57   GiB
        Target Size: (Adaptive)         79.89%  25.57   GiB
        Min Size (Hard Limit):          12.50%  4.00    GiB
        Max Size (High Water):          8:1     32.00   GiB

ARC Efficiency:                                 1.25b
        Cache Hit Ratio:                90.52%  1.13b
        Cache Miss Ratio:               9.48%   118.08m
        Actual Hit Ratio:               84.54%  1.05b

        Data Demand Efficiency:         95.45%  356.90m
        Data Prefetch Efficiency:       40.64%  11.36m

L2 ARC Breakdown:                               118.18m
        Hit Ratio:                      62.87%  74.29m
        Miss Ratio:                     37.13%  43.89m
        Feeds:                                  849.64k

File-Level Prefetch: (HEALTHY)

DMU Efficiency:                                 28.09b
        Hit Ratio:                      88.54%  24.87b
        Miss Ratio:                     11.46%  3.22b
```

## zfs-stats and zfs-mon: ZFS Statistics Tools

The `kstat.zfs` sysctl(8) knobs provide access to many ZFS counter variables. These variables contain raw data and to make any conclusions from these variables, intermediate values need to be computed. The perl scripts zfs-stats and zfs-mon do process this data and provide human-readable output. The zfs-stats tool is based on Ben Rockwood's `arc_summary.pl` and includes modifications by Jason J. Hellenthal and myself. Both tools are available in the FreeBSD ports tree as `sysutils/zfs-stats`. The data in zfs-stats summarizes and/or averages counters that collect data since the system was booted. Example output excerpt from zfs-stats: Listing 1.

The uptime averages do not tell much about the actual system performance. To view real-time cache efficiency (or raw numbers) I have written the "zfs-mon" tool. It monitors ARC, L2ARC and zfetch in real time and outputs 10s, 60s and total per second averages (total = since the program was started).

Example "zfs-mon -a" output after collecting 120 seconds of data: Listing 2.

As of total cache efficiency, the L2ARC cache is accessed only on an ARC miss, so your total cache efficiency is calculated using the following formula: `[ARC efficiency] + (100-[ARC efficiency])*([L2ARC efficiency]/100)`.

Result for the example above: `89,96 + (100-89,96)*(71,15/100) = 97,10`.

### Interpreting the output from zfs-stats and zfs-mon

The output of zfs-stats and zfs-mon may help you to discover bottlenecks and decide to change some de-

fault values. The main values to look at are usage and efficiency of various caches. An efficiency value of 100% means all reads are done from the cache, 0% means all reads are done from disks. For my uses, efficiency above 80% counts as efficient and above 90% counts as highly efficient. Remember, that L2ARC takes some time to warm-up and is intended to improve your total cache efficiency. If using zfs-mon, try to collect data for a longer period of time and watch the "tot" column.

**Here are some general tips:**
Inefficient ARC data cache:

- if you have limited the ARC size, increase or remove the limit
- disable ARC for some datasets

**Listing 2.** *Example output of "zfs-mon -a" (runtime 120 seconds)*

```
ZFS real-time cache activity monitor
Seconds elapsed: 120

Cache hits and misses:
                                1s    10s    60s    tot
                     ARC hits:   259    431    418    466
                   ARC misses:    51     40     49     52
         ARC demand data hits:   223    417    390    437
       ARC demand data misses:    36     20     17     16
     ARC demand metadata hits:    36     11     25     25
   ARC demand metadata misses:    15     19     21     25
       ARC prefetch data hits:     0      4      3      4
     ARC prefetch data misses:     0      1     10      8
   ARC prefetch metadata hits:     0      0      0      0
 ARC prefetch metadata misses:     0      0      1      3
                  L2ARC hits:    47     34     40     37
                L2ARC misses:     4      5      9     15
                ZFETCH hits: 47903  47294  48155  47138
              ZFETCH misses:   272    449   1147   3593

Cache efficiency percentage:
                          10s    60s    tot
                 ARC:   91.51  89.51  89.96
       ARC demand data:  95.42  95.82  96.47

     ARC prefetch data:  80.00  23.08  33.33
ARC prefetch metadata:    0.00   0.00   0.00
               L2ARC:   87.18  81.63  71.15
              ZFETCH:   99.06  97.67  92.92
```

- considering lowering the ARC metadata limit
- add more RAM to your system
- consider using additional L2ARC cache devices

Inefficient ARC metadata cache:

- consider increasing the ARC metadata limit
- add more RAM to your system

Inefficient L2ARC cache:

- this depends very much on the structure of your reads
- if your ARC is already very efficient, L2ARC might sometimes add only little advantage
- if your ARC is inefficient, too, consider increasing system memory and L2ARC
- in some scenarios L2ARC efficiency of 30-40% may already be acceptable

Inefficient ZFETCH:

- consider disabling zfetch

Inefficient vdev prefetch:

- consider disabling vdev prefetch
- if enabled, scrub and resilver may run substantially faster
- modify advanced vdev prefetch settings (experts only)

## Tuning ZFS for Applications

### Webservers

On FreeBSD, user experience has shown that it is an advantage to disable sendfile and mmap on your webservers, if you are serving your pages from ZFS datasets. Otherwise your data may get cached in your memory twice and this reduces your system memory.

Here are example configuration directives for popular webservers:

**Apache**

```
EnableMMAP Off
EnableSendfile Off
```

**Nginx**

```
Sendfile off
```

**Lighttpd**

```
server.network-backend = "writev"
```

### Database servers

For databases like PostgreSQL and MySQL, users recommend to store them on a dataset created with a different recordsize than the default of 128 kilobytes.

For PostgreSQL and MySQL (MyISAM storage), set the recordsize to 8 kilobytes before populating the dataset:

```
# zfs create -o recordsize=8k tank/mysql
```

For MySQL InnoDB storage files (not logs) set the recordsize to 16 kilobytes, logs might be left at the default record size (you need to spread the data over more datasets that have different recordsizes).

### NFS servers

If sharing ZFS datasets on NFS servers with a lot of writes, the *ZFS Intent Log* (ZIL) might be your bottleneck. To improve performance, you may want to move ZIL to separate log devices (fast SSD drives) or try to disable ZIL for the affected datasets. Disabling ZIL may cause NFS client corruption.

```
# zfs set sync=disabled dataset
```

## ZFS and 4k Sector Drives

Originally hard drives used to store data in 512 byte physical sectors. Today's large harddrives use the Advanced Format (4k sectors) and provide a 512 byte-sector compatibility mode. On FreeBSD, the mentioned drives usually still report a 512 byte sector size:

```
ada0: 2861588MB (5860533168 512 byte sectors: 16H 63S/T 16383C)
```

ZFS uses this value when defining the block size for devices when added to a pool (or when a pool is created). To view your pool device configuration, use the following command:

```
# zdb -C [poolname]
```

The parameter "ashift" describes the ZFS block size used size as 2 ^ [ashift]. A value of "9" means 512 byte sectors. To have 4 kilobyte blocks, a value of "12" is required.

Users report poor performance with Advanced Format hard drives and ahift=9, especially in raidz configurations. To create a pool optimized for 4 kbyte sectors, we have to have to make the ZFS block size match the physical sector size do some tricks with a fake gnop device. Let's assume we want to create a new pool with the `/dev/ada0` device:

### Tuning links

- *http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide*

### Technical links

- *http://dtrace.org/blogs/brendan/2012/01/09/activity-of-the-zfs-arc*
- *http://dtrace.org/blogs/brendan/2008/07/22/zfs-l2arc*
- *https://blogs.oracle.com/roch/entry/tuning_zfs_recordsize*
- *https://blogs.oracle.com/roch/entry/dedup_performance_considerations1*
- *http://constantin.glez.de/blog/2011/02/frequently-asked-questions-about-flash-memory-ssds-and-zfs*
- *http://ivoras.net/blog/tree/2011-01-01.freebsd-on-4k-sector-drives.html*

```
# gnop create -S 4096 ada0
# zpool create tank ada0.nop
# zpool export tank
# gnop destroy ada0.nop
# zpool import tank
# zdb -C tank | grep ashift
```

On a 4k block setup, small files up to 4k always take one whole block. ZFS metadata is many times smaller than 4kb. Please consider that using ashift=12 increases the initial space required for metadata by a fairly large amount (about 5% of your total disk space). Depending on your data, this overhead may increase on filling the pool with data (e.g. many small files). So this is effectively a tradeoff between performance and free space and you have to decide which is more important.

## Conclusion

Default values of ZFS settings are intended to suit the average user. This article presented several ways how to optimize a system for ZFS and how to tune these values for specific workloads. The evaluation tools zfs-stats and zfs-mon provide necessary measurement data.

ZFS is a great piece of software and I use it heavily on dozens of systems in combination with FreeBSD, OpenIndiana and even Linux. The lack of measurement and evaluation tools have inspired me to work on zfs-stats and zfs-mon.

**MARTIN MATUŠKA**

***Martin Matuška (mm@FreeBSD.org) is an IT expert, senior systems administrator and developer. He is part of the FreeBSD ZFS team, maintainer of several FreeBSD ports and head of the system administration company VX Solutions s. r. o. (http://www.vx.sk). His company focuses on deploying and maintaining ZFS systems and providing solutions based on FreeBSD, Linux and IllumOS operating systems. He writes at http://blog.vx.sk***

# MPD5

## VPN Server with FreeBSD Setup and Management

Mpd5 is a fast, flexible and secure way to make VPN connections on FreeBSD. It requires very few resources and supports a wide range of protocols, a great tool for network managers.

### What you will learn…

- In this paper we will learn to setup and manage a VPN server PPTP based.

### What you should know…

- Basics about how to compile BSD Kernel and basic BSD Networking Setup

The package we will use is mpd5 and is particularly suitable for those who need to grant VPN access to external consultants, the so-called "Road Warriors". The operating system is FreeBSD version 8.2 release, but the configuration is also valid for version 9.x. The internal network will use the address family 10.0.0.0/255.255.255.0 and the internal address of the BSD firewall will be 10.0.0.1.

The goal is to setup and manage a number of connection links between the PPTP VPN server and external users, simultaneously, by using the Netgraph implementation of *Point-to-Point Tunneling Protocol* (PPTP), a sys-

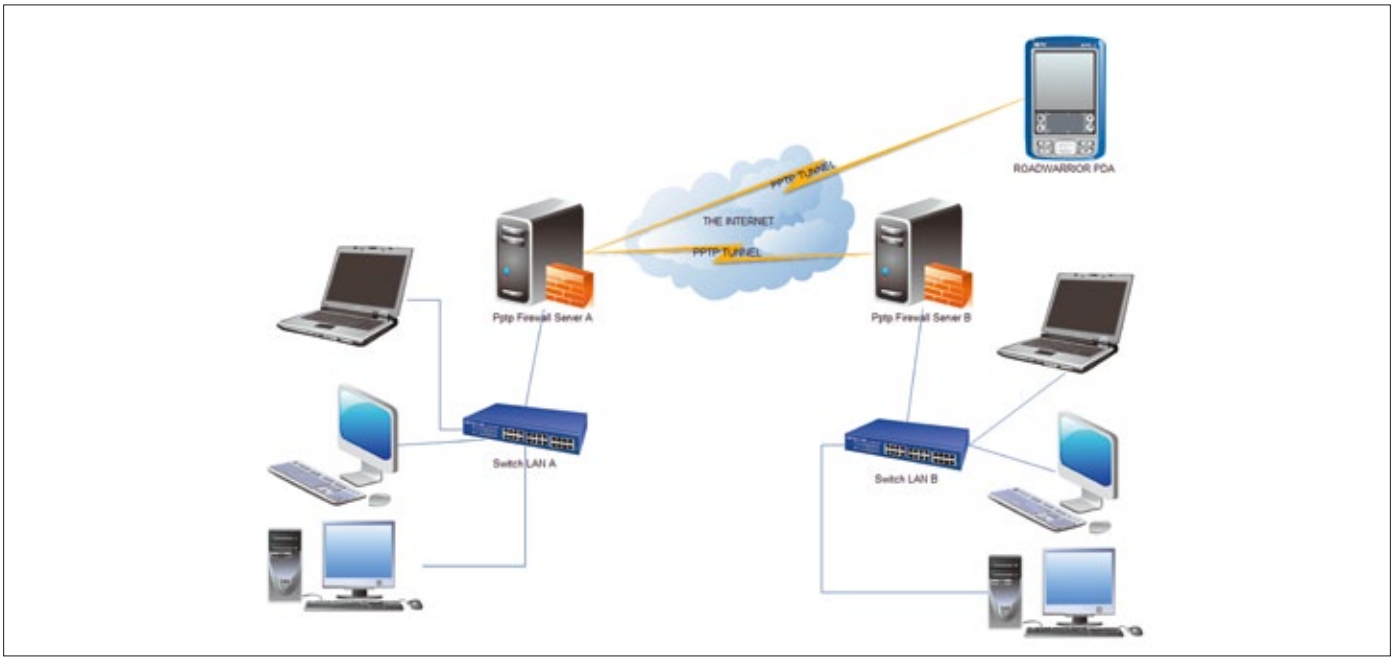



**Figure 1.** *Mpd5 VPN PPTP connections examples*

tem used to implement virtual private networks that uses a control channel over TCP and a GRE tunnel operating to encapsulate PPP packets.

## What is the Netgraph System

Netgraph is the graph based kernel networking subsystem and provides an uniform and modular environment to implement kernel objects which perform various networking functions, known as nodes. The nodes can be assembled in particular structures, so-called graphs, whose edges are formed by the nodes "hooks" and are used to manage data flows. With netgraph one has a flexible and modular implementation to manage connections and both protocol and network layer, using a fast kernel based architecture. Mpd5 can serve several Gigabits per second of PPP traffic with right hardware, allowing many thousands of simultaneous connections without important performance degradation, moreover it needs very low resources, finally it supports many of existing PPP link types like modem for asychronous serial connections, pptp for the *Point-to-Point Tunnelling Protocol* (PPTP), l2tp for *Layer Two Tunnelling Protocol* (L2TP), pppoe for Ethernet connections with the *PPP-over-Ethernet* (PPPoE) protocol, *tcp/udp* to tunnel PPP session over a TCP/UDP connection. With different configurations mpd5 is able to run as a PPP client or server and supports many PPP protocols extensions, such as MS-CHAP and EAP authentication, it uses traffic encryption like *MPPE* and includes many additional features like *Network address translation* (NAT) support, telnet and http control interfaces, IPv4 and IPv6 protocols support, and different authentication and accounting methods support such as RADIUS, PAM, script, or file.

## Installing mpd5

One can install mpd5 using ports or binay packages

```
cd /usr/ports/net/mpd5
make install

pkg_add -rv mpd5
```

**Note**

Mpd5 documentation is installed in HTML and PostScript format into `/usr/local/share/doc/mpd`.

## Configuring mpd5

The mpd5 configuration folder is in `/usr/local/etc/mpd5`, and its main configuration file is `mpd.conf`. For the network previously shown, here is an example configuration file: Listing 1.

**Listing 1.** *The mpd5 core configuration file*

```
root@freeBsd82-fw.virtual.test/usr/local/etc/mpd5#cat
                    mpd.conf
#################################################
#                 MPD configuration file              #
#################################################
startup:
# Disabling web and console access
set console close
set web close

default:
        load pptp_server

pptp_server:
        set ippool add poolsat 10.0.0.200 10.0.0.220
        create bundle template B
        set iface enable proxy-arp
        set iface idle 0
        set iface enable tcpmssfix
        set ipcp yes vjcomp
        set ipcp ranges 10.0.0.254/32 ippool poolsat
        set ipcp dns 10.0.0.1

# Enabling Microsoft Point-to-Point encryption (MPPE)
        set bundle enable compression
        set ccp yes mppc
        set mppc yes compress e40 e56 e128 stateless

# Creating clonable link template named L
        create link template L pptp
# Setting bundle template to use
        set link action bundle B
# Using Multilink for giving full 1500 MTU
        set link enable multilink
        set link yes acfcomp protocomp
        set link no pap chap eap
        set link enable chap
        set link enable chap-msv2
# Reducing link mtu to avoid GRE packet fragmentation.
        set link mtu 1440
        set link keep-alive 10 60

# Configuring PPTP and open link
        set pptp self 0.0.0.0 # So configured,daemon
                    listens on all interfaces
# Allowing to accept calls
        set link enable incoming
```

In the above example configuration, we specified the following fundamental settings:

- `set ipcp ranges <mpd-ip-address>/<mask> ippool poolsat`: the ip address that our mpd5 daemon is listening on, in the subnet mask of our internal network.
- `set ipcp dns <dns-server>`: the DNS server IP address.
- `set pptp self <mpd-ip-address>`: Another time this is the ip address on which our mpd5 daemon is listening on
- `set ippool add poolstat <ip-range1> <ip-range2>`: the IP Pool reserved for PPTP connections.

Now we must add user accounts that will be authorized to connect to our PPTP server, they are stored in the `/usr/local/etc/mpd5/mpd.secret` file. This file contains username-passwords pairs that are used to authorize a user, separated by a whitespace, a particular use provides a fixed ip address to a particular username,

**Listing 2.** *MPD credentials configuration file*

```
#####################################################
#            MPD secrets configuration file         #
#####################################################
user_2012  user_2012
user2_2012 user2_2012 10.0.0.222

# An external password access program
user3_2012     "!/usr/local/etc/mpd5/passwd_get.sh"
```

**Listing 3.** *An mpd basic firewall ruleset to add to /usr/local/etc/ipfw.rules*

```
## ipfw firewall ruleset for mpd

# Macros
fwcmd=/sbin/ipfw

# Allow Mpd pptp traffic
$fwcmd add 0554 allow tcp from any to any dst-port 1723
$fwcmd add 0555 allow tcp from any 1723 to any
$fwcmd add 0556 allow gre from any to any

# MPD Vpn Rulesets
$fwcmd add 0671 allow gre from any to any
$fwcmd add 0672 allow all from any to any via ng0
$fwcmd add 0673 allow all from any to any via ng1
$fwcmd add 0674 allow all from any to any via ng2
$fwcmd add 0675 allow all from any to any via ng3
$fwcmd add 0676 allow all from any to any via ng4
$fwcmd add 0677 allow all from any to any via ng5
$fwcmd add 0678 allow all from any to any via ng6
$fwcmd add 0679 allow all from any to any via ng7
$fwcmd add 0680 allow all from any to any via ng8
```

**Listing 4.** *The FreeBSD Netgraph Kernel Modules*

```
root@freeBsd82-fw.virtual.test/home/utente#kldstat
Id Refs Address    Size     Name
 1    9 0xc0400000 c67484   kernel
 2    1 0xc3133000 4000     ng_socket.ko
 3    3 0xc3137000 b000     netgraph.ko
 4    1 0xc31cd000 4000     ng_iface.ko
 5    1 0xc31e2000 7000     ng_ppp.ko
```

**Listing 5.** *The FreeBSD Netgraph Kernel Configuration Options*

```
# NETGRAPH KERNEL SUBSYSTEM

options         NETGRAPH                #netgraph(4) system
options          NETGRAPH_ASYNC
options          NETGRAPH_BPF

options          NETGRAPH_ECHO
options          NETGRAPH_ETHER
options          NETGRAPH_FRAME_RELAY
options          NETGRAPH_HOLE
options          NETGRAPH_IFACE
options          NETGRAPH_KSOCKET
options          NETGRAPH_L2TP
options          NETGRAPH_LMI

# MPPC compression requires proprietary files (not included)

#options         NETGRAPH_MPPC_COMPRESSION
options          NETGRAPH_MPPC_ENCRYPTION
options          NETGRAPH_ONE2MANY
options          NETGRAPH_PPP
options          NETGRAPH_PPPOE
options          NETGRAPH_PPTPGRE
options          NETGRAPH_RFC1490
options          NETGRAPH_SOCKET
options          NETGRAPH_TEE
options          NETGRAPH_TTY
options          NETGRAPH_UI
options          NETGRAPH_VJC
```

the following is a good example: Listing 2. The starting "!" means that the password for user user3_2012 is not stored in the mpd.secret file directly, but will be get by using the command `/usr/local/etc/mpd5/passwd_get.sh user3_2012`, for example obtaining it from a database. This system allows sysadmin to print the plaintext password for the named user as a single line to standard output, and then exit, allowing mpd5 to intercept this operation. If there is an error, the command should print what is the matter, helping in troubles resolution.

Another special case is the username "*" in the mpd.secret file, particularly this line must be the last of the file and will match any username,using external programs like the previous `passwd_get.sh` to check a valid username. This wildcard matching only works for "!" lines.

It's basic that the total length of the executed command is less than 128 characters. Finally, it's important to know that any additional arguments generated by the scripts will be visible to users on the local machine running `ps`.

A good security practice is to limit access to the configuration folder as follows:

```
# chown -R root:root /usr/local/etc/mpd5
# chmod -R 0600 /usr/local/etc/mpd5
```

## Allowing mpd5 Traffic Through your Firewall with ipfw

The standard firewalling system used in FreeBSD is `ipfw`. If we are using mpd5 in a firewalled environment, we must pass the traffic through the firewall for permitting clients to connect to the PPTP server and our internal network, and this is a real example: Listing 3.

## Starting mpd5

To allow mpd5 daemon to start at boot time, we must add these lines to our `/etc/rc.conf` file:

```
# Host acts like a router
gateway_enable="YES"

# MPD5
mpd_enable="YES"
mpd_flags="-b -s mpd5"
arpproxy_all="YES"
```

And then restart network subsystem and mpd5 daemon, so our clients will be able to connect to it.

```
# /etc/netstart
# /usr/local/etc/rc.d/mpd5 start
```

### On the Net

- Netgraph: *http://www.freebsd.org/cgi/man.cgi?query=netgraph&sektion=4*
- Mpd5 Documentation: *http://mpd.sourceforge.net/**doc5/*
- *http://mpd.sourceforge.net/doc5/*
- FreeBSD kernel compiling: *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig-building.html*

The final step is to enable mpd5 logging with syslog, and we can do this by creating a new empty file and changing its ownership only for root user

```
# touch /var/log/ppp.log
# chmod 600 /var/log/ppp.log
```

Then we must add these lines in `/etc/syslog.conf`

```
!mpd
*.*     /var/log/ppp.log
```

After we saved the file, finally we must restart syslog

```
# /etc/rc.d/syslogd reload
```

**Note**

We should check that all needed modules are loaded before putting the server online (Listing 4).

## More Info About netgraph Subsystem

Many user-space applications use the Netgraph facility as well as mpd5, and here is a list of netgraph kernel configuration options for various device-independent node types (Listing 5).

## Conclusion

Mpd5 is a great way to establish secure connections reasonably affordable with any compatible device (IPAD, Android and of course, laptops and PC), all in a simple and fast way ,really a "must to try".

**ANTONIO FRANCESCO GENTILE**

***Antonio Francesco Gentile lives in Italy, Calabria, is a software and network engineer. He works for a company in Rome as network manager, with the "Culture Lab" http://culture.deis.unical.it Department of Telematics at University of Calabria, the computer science associations "Hacklab Cosenza" http://hacklab.cosenzainrete.it/ and "Verde Binario" http://www.verdebinario.org/ and is a freelance columnist for Italian magazines "Linux&C" http://www.oltrelinux.com/ and "Linux Magazine" http://www.linux-magazine.it/.***

# PostgreSQL

## Partitioning (part 1)

In the previous articles the main features of PostgreSQL, including server-side programming were shown. In this article a simple application scenario will be used to demonstrate the capability of partitioning huge amounts of data into different tables in different spaces transparently.

**What you will learn…**

- Basic concepts of data partitioning
- how to implement a partition using PostgreSQL features
- table inheritance

**What you should know…**

- basic shell commands
- basic PostgreSQL concepts
- server-side programming concepts as explained in the previous papers

Please note that the examples shown here, in order to be as realistic as possible, make use of a few hundred megabytes of data and will require a bit of time to execut and will consume part of your disk bandwidth. All the examples have been tested on a PostgreSQL 9.1 cluster running on a FreeBSD 8.2-RELEASE machine.

### Data Partitioning

Data partitioning is a technique that aims at providing better performances and modularity, splitting the data set into groups with similar features. Partitioning makes sense when the amount of data becomes big and/or complex and there is a either a need for performance improvement or a storage requirement. To put it simply, splitting the data set into smaller groups allows for both faster archiving of each group and faster data scanning (i.e., queries on the data).

The partitioning is done in an horizontal way, meaning that data is not split into joined tables, but into tables that group data depending on some of their attribute values. Such tables will therefore have the same set of attributes, but different values which will determine the table that will handle a set of tuples. Since each table has a "fixed" set of values, indexes and queries generally will not need to filter such attributes, resulting in a smaller and faster data search path.

### Table Inheritance

PostgreSQL supports table inheritance. That means a table can inherit (i.e., extend) a parent table. The concept is pretty similar to the OOP inheritance: having defined a master (or parent) table, it is possible to define a table that has the same properties (i.e., columns) and provides extensions (e.g., other columns, constraints, and so on) to the parent set. When applying table inheritance it is important to remember a few concepts:

- `INSERT` statements are routed automatically to the parent table and not to the children;
- other SQL statements (such as `SELECT`, `UPDATE`, `DELETE`) automatically provide a union of the parent table and all its children, unless told to do not do so;
- there is no automatic way to apply an unique constraint (like a key) over a parent-child table: each table is considered as a different entity and therefore with different constraints;
- foreign keys and unique constraints are not inherited and have to be set-up manually.

Table inheritance is a key feature for data partitioning, since it guarantees that all partitions (i.e., child tables that inherit from a common ancestor) have the same data structure, allowing the DBA to define the conditions that make the each partitions different from the others.

## Application Scenario

Partitioning is worth applying to a large database with datasets that can be split depending on a few well defined criteria. As an example, a database for a simple Web forum will be implemented. Since Web forums are commonly used tools with a very high average of messages per user; they can be considered a good test case for a partitioned scenario. As usual, the example presented here is kept as simple as possible for didactic purposes.

The first step is to create a new database, named `forumdb`, and its superuser `forum` that will handle all the forum related data. In order to do that, connect as `pgsql` to the `template1` database and issue the following commands:

```
template1=# CREATE USER forum WITH LOGIN ENCRYPTED
                    PASSWORD 'forum';
CREATE ROLE
template1=# CREATE DATABASE forumdb WITH OWNER forum;
CREATE DATABASE
```

It is now possible to connect to the new database and to define the structure and the tables as shown in Listing 3. For the purposes of this example it suffices to have an *author* table that will contain information about each single forum user, a *category* table that will contain a description of available forum categories (e.g., hardware, general, networking and so on) and a table to contain each message (*thread*). The *thread* table is the core of the forum database and contains a message title and content, as well as foreign keys that reference the message's the author and its category. To keep the example simple each message is identified by a "thread identifier" and a "message identifier", columns *tid* and *mid*, respectively. All messages with the same *tid* belong to the same thread and are ordered by the *mid* value, with the starting thread having a mid value of zero.

The view *vw_thread* provides an easy way to get a list of all threads with information about authors, categories

**Listing 1.** *Output of vw_thread view*

```
forumdb=> SELECT * FROM vw_thread WHERE thread_id = 1;
thread_id | category |   main_title   |        started_by     |       message      |         replied_by
----------+----------+----------------+-----------------------+--------------------+----------------------------
        1 | Hardware | Thread start 1 | Luca Ferrari @ 1991-01-11 08:02:00 | How do you do this? |
        1 | Hardware |                |                       | I think so and so.  | A. Ferrari @ 1991-01-11 08:04:00
        1 | Hardware |                |                       | I think so and so. | Claudio @ 1991-01-11 08:06:00
        1 | Hardware |                |                       | I think so and so. | Ritchie @ 1991-01-11 08:08:00
        1 | Hardware |                |                       | I think so and so.  | A. Ferrari @ 1991-01-11 08:10:00
        1 | Hardware |                |                       | I think so and so. | Claudio @ 1991-01-11 08:12:00
        1 | Hardware |                |                       | I think so and so. | Ritchie @ 1991-01-11 08:14:00
        1 | Hardware |                |                       | I think so and so.  | A. Ferrari @ 1991-01-11 08:16:00
        1 | Hardware |                |                       | I think so and so. | Claudio @ 1991-01-11 08:18:00
        1 | Hardware |                |                       | I think so an
```

**Listing 2.** *Size of the thread table*

```
forumdb=> SELECT c.description, COUNT(t.mid), MIN(t.published_on), MAX(t.published_on)
FROM thread t JOIN category c ON t.category_pk = c.pk
GROUP BY c.description
ORDER BY c.description;
 description |  count  |    min     |    max
-------------+---------+------------+------------
 General     | 1310540 | 1990-01-05 | 2012-04-17
 Hardware    |  391200 | 1991-01-11 | 2004-03-01
 Kernel      |  848802 | 1993-01-05 | 2012-04-17
 Networking  | 1493115 | 1992-01-05 | 2012-04-17
```

**Listing 3.** *The definition of the database for the forum application scenario*

```
DROP TABLE IF EXISTS author;
CREATE TABLE author(
       pk SERIAL NOT NULL,
       username  text NOT NULL,
       display_name text,

       PRIMARY KEY(pk),
       UNIQUE( username )
);

INSERT INTO author( username, display_name)
VALUES( 'fluca1978', 'Luca Ferrari' );

INSERT INTO author( username, display_name)
VALUES( 'ringhiobd', 'A. Ferrari' );

INSERT INTO author( username, display_name)
VALUES( 'winter', 'Claudio' );

INSERT INTO author( username, display_name)
VALUES( 'root', 'Ritchie' );


DROP TABLE IF EXISTS category;
CREATE TABLE category(
       pk serial NOT NULL,
       id varchar NOT NULL,
       description text,
       since date default 'now'::text::date,

       PRIMARY KEY(pk),
       UNIQUE(id)
);

INSERT INTO category(id, description, since)
VALUES('misc', 'General', '01/01/1990'::text::date);
INSERT INTO category(id, description, since)
VALUES('net', 'Networking', '01/01/1992'::text::date);
INSERT INTO category(id, description, since)
VALUES('kern', 'Kernel','01/01/1993'::text::date);
INSERT INTO category(id, description, since)
VALUES('dev', 'Development','31/01/1993'::text::date);
INSERT INTO category(id, description, since)
VALUES('hw', 'Hardware', '01/07/1991'::text::date);


DROP TABLE IF EXISTS thread;
CREATE TABLE thread(
       pk serial   NOT NULL,
       tid integer NOT NULL,
       mid integer NOT NULL,
       title text,
       content text,
       published_on date default 'now'::text::date,
       published_at time default 'now'::text::time,
       category_pk integer,
       author_pk   integer,

       PRIMARY KEY(pk),
       UNIQUE(tid, mid),
       FOREIGN KEY(category_pk) REFERENCES category(pk),
       FOREIGN KEY(author_pk)   REFERENCES author(pk)
);



CREATE OR REPLACE VIEW vw_thread
AS
SELECT tid AS thread_id,
       c.description as category,
       CASE WHEN t.mid = 0 THEN title
            ELSE ''
       END as main_title,
       CASE WHEN t.mid = 0 THEN a.display_name || '
                  @ ' || t.published_on || ' ' ||
                  t.published_at
            ELSE ''
       END as started_by,
       t.content as message,
       CASE WHEN t.mid <> 0 THEN a.display_name || '
                  @ ' || t.published_on || ' ' ||
                  t.published_at
            ELSE ''
       END as replied_by
FROM ((thread t JOIN author a ON author_pk = a.pk) JOIN
                  category c ON t.category_pk = c.pk)
ORDER BY t.tid, t.mid;
```

**Listing 4a.** *The populate_forum stored procedure*

```
CREATE OR REPLACE FUNCTION populate_forum()
RETURNS VOID
AS
$BODY$
DECLARE
        category_multiplier     integer;
        thread_per_author       integer;
        message_per_thread      integer;
        available_authors       integer;
        current_tid             integer;
        current_mid             integer;
        current_category        category%rowtype;
        current_author          author%rowtype;
        current_writer          author%rowtype;
        current_published_at    time;
        current_published_on    date;

BEGIN

        -- a multiplier for the size of each category
        category_multiplier := 1;

        -- how many authors are there?
        SELECT count(pk)
        FROM   author
        INTO   available_authors;

        -- append to the max tid if any is existing
        SELECT max(tid)
        FROM   thread


        IF current_tid IS NULL THEN
           current_tid := 0;
        ELSE
           current_tid := current_tid + 1;
        END IF;

        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP
                -- increment the category multiplier
                category_multiplier := category_
                    multiplier + 1;

                RAISE INFO 'Category % (multiplier
                    %)', current_category.description,
                    category_multiplier;


                -- set the initial date and time for
                    publishing this category
                current_published_on := current_
                    category.since + 2;
                current_published_at := time '08:02:00';
                RAISE INFO 'category start date % at
                    time %', current_published_on,
                    current_published_at;


                -- iterate on each author
                FOR current_author IN SELECT *
                                      FROM   author
                                      ORDER BY username
                                      LOOP

                   -- how many message per author
                and per thread?
                   thread_per_author  := category_
                multiplier * 300;


                  WHILE thread_per_author > 0 AND
                current_published_on < current_date LOOP
                          --RAISE INFO ' % threads
                remaining', thread_per_author;
                        current_tid := current_tid + 1;
                          current_mid := 0;
                       current_published_on := current_
                published_on + interval '2 days';

                         INSERT INTO thread( tid, mid,
                title, content, category_pk, author_pk,
                published_on, published_at )
                          VALUES( current_tid,
                current_mid, 'Thread start ' ||
                current_tid, 'How do you do this?',
                current_category.pk, current_author.
                pk, current_published_on, current_
                published_at );
```

**Listing 4b.** *The populate_forum stored procedure*

```
            thread_per_author :=
thread_per_author - 1;
            message_per_thread :=
category_multiplier * 20 * available_
authors;


        WHILE message_per_thread >
0 LOOP
            -- now insert replies from
other authors
            FOR current_writer IN
SELECT *
                                FROM
author
                                WHERE
pk <> current_author.pk
                                ORDER
BY display_name
                                LOOP
                current_mid :=
current_mid + 1;
                message_per_thread :=
message_per_thread - 1;
                current_published_at
:= current_published_at + interval '2
minutes';
                INSERT INTO thread(
tid, mid, title, content, category_
pk, author_pk, published_on,
published_at )
                VALUES( current_tid,
current_mid, 'Thread reply ' ||
current_tid, 'I think so and so.',
current_category.pk, current_writer.
pk, current_published_on, current_
published_at );

            END LOOP;
        END LOOP;  -- end of the while
for the thread messages

    END LOOP; -- end of the while per
author
END LOOP;  -- end of the iteration over
authors
        END LOOP;       -- end of the category iteration


END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 5.** *A stored procedure that creates all the per-category tables*

```
CREATE OR REPLACE FUNCTION create_category_tables()
RETURNS integer
AS
$BODY$
DECLARE
        current_category        category%rowtype;
        created_tables          integer;
BEGIN

        created_tables := 0;
        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP


                -- a dynamic query for creating the table
                 EXECUTE 'CREATE TABLE IF NOT EXISTS
                    thread_' || current_category.id
                        || ' ( CHECK(category_pk = ' ||
                    current_category.pk || '), '
                        || ' PRIMARY KEY(pk), '
                        || ' FOREIGN KEY(category_pk)
                    REFERENCES category(pk), '
                        || ' FOREIGN KEY(author_pk)
                    REFERENCES author(pk), '
                        || ' UNIQUE(tid, mid) '
                        || ') INHERITS (thread);';
                created_tables := created_tables + 1;
        END LOOP;       -- end of the category iteration
        RETURN created_tables;
END;
$BODY$
LANGUAGE plpgsql;
```

and thread start/replies, so that for instance two different threads are shown as follows: Listing 1.

Having defined the main structures, it is possible to populate the database with a set of threads and replies. In order to simulate a workload the special stored procedure *populate_forum* has been defined (see Listing 4). The idea is that for each category each author will start a certain number of threads and all the other authors will post a specific number of replies to those threads; by convention each thread starts two days after the previous one from the same author, and all replies come with a little different time within the same day; finally the population stops when either the number of threads-per-author or the current date is reached on each category. While the time scenario is not a real one, it will be useful to stress the partitioning implementation.

Executing the stored procedure makes the *thread* table grow to more than 4 million tuples, scattered amongst categories as follows: Listing 2.

For a total number of 4043657 tuples in 49828 data pages for around 390 MB of disk space. It is now time to partition the data!

## Partitioning Using the Forum Category

A first kind of partitioning could be based on the message category: since each thread belongs to one and only one category and each category can live on its own, this is a good first partition schema. In order to implement this partition it is required to:

- create a table for each category; such table will handle all the threads that belong to such category. It is worth noting that these tables have the same data structure of the main *thread* table;
- build appropriate constraints on each table to avoid the erroneous or malicious manipulation of data belonging to another category (e.g., the "net" category table cannot accept threads or queries that refers to the "kern" category);

**Listing 6.** *A stored procedure that migrates all the data into the appropriate table*

```
CREATE OR REPLACE FUNCTION migrate_threads()
RETURNS VOID
AS
$BODY$
DECLARE
        current_category        category%rowtype;
BEGIN

        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP


                RAISE INFO 'Inserting for category %',
                    current_category.description;
                -- copy each tuple in the right table
                EXECUTE 'INSERT INTO thread_' ||
                    current_category.id
                         || ' SELECT * FROM thread '
                         || ' WHERE category_pk = ' ||
                    current_category.pk;

                -- delete tuples from the master table
                RAISE INFO 'Deleting for category %',
                                    current_category.description;
                EXECUTE 'DELETE FROM ONLY thread '
                         || ' WHERE category_pk = ' ||
                    current_category.pk;

        END LOOP;       -- end of the category iteration

END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 7.** *A rule to handle inserting into the net category*

```
CREATE OR REPLACE r_thread_insert_net AS
    ON INSERT TO thread
   WHERE new.category_pk = 2
   DO INSTEAD
   INSERT INTO thread_net (pk, tid, mid, title, content,
                           published_on, published_at,
                   category_pk,
                           author_pk)
   SELECT new.pk, new.tid, new.mid, new.title, new.
                   content,
                           new.published_on, new.
                   published_at, new.category_pk,
                           new.author_pk
```

**Listing 8.** *A stored procedure to automate the creation of all rules associated to the thread table*

```
CREATE OR REPLACE FUNCTION create_category_rules()
RETURNS integer
AS
$BODY$
DECLARE
        current_category        category%rowtype;
        created_rules           integer;
BEGIN
        created_rules := 0;
        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP

                -- rule for INSERT
                EXECUTE 'CREATE OR REPLACE RULE r_thread_
                    insert_' || current_category.id
                        || ' AS ON INSERT TO thread '
                        || ' WHERE NEW.category_pk = '
                    || current_category.pk
                        || ' DO INSTEAD '
                        || ' INSERT INTO thread_' ||
                    current_category.id
                        || ' SELECT NEW.*';

                created_rules := created_rules + 1;

        END LOOP;       -- end of the category iteration

        RETURN created_rules;
END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 9.** *A trigger function that re-routes tuple insertion*

```
CREATE OR REPLACE FUNCTION thread_partitioning_handler()
RETURNS trigger
AS
$BODY$
DECLARE
        current_category_id             text;
        current_category_table_name     text;
BEGIN
        RAISE LOG 'Trigger executing as %', TG_OP;
        -- get the name of the category
        IF TG_OP = 'INSERT' THEN
                SELECT id
                FROM   category
                INTO   current_category_id
                WHERE  pk = NEW.category_pk;
        END IF;

        -- build the table name for the right insert
        SELECT 'thread_' || current_category_id
        INTO   current_category_table_name;

        RAISE LOG 'The query is going to be re-routed to
                    the table %',
                        current_category_table_name;

        -- execute the insertion into the right table
        IF TG_OP = 'INSERT' THEN

                EXECUTE 'INSERT INTO '
                        || current_category_table_name
                        || ' SELECT $1.* '
                        USING NEW;
        END IF;

        RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 10.** *A stored procedure to create all the re-routing triggers*

```
CREATE OR REPLACE FUNCTION create_category_triggers()
RETURNS void
AS
$BODY$
DECLARE
        current_category        category%rowtype;
BEGIN
        EXECUTE 'DROP TRIGGER IF EXISTS tr_thread_
                    trigger ON thread';

                || ' BEFORE INSERT '
                || ' ON thread '
                || ' FOR EACH ROW '
                || ' EXECUTE PROCEDURE thread_
                    partitioning_handler()';
END;
$BODY$
LANGUAGE plpgsql;
```

- migrate all existing data from the *thread* table to the right per-category table;
- avoid the direct manipulation of the *thread* table, since all the queries must be "routed" to a specific per-category table;
- allow transparent querying of the *thread* table and of its per-category children.

### Steps 1 and 2: Creating the Tables and the Constraints

The first two steps can be done using a stored procedure, so that if the number of categories changes over time new partition tables can be created in an automated way. The idea is simple: for each *category* in the category table a new table named *thread_categoryIdentifier* (e.g., *thread_kern*) will be created. Such table will have the same data structure of the main thread table and will have a check constraint that assures that the *category_pk* column refers to only one category; moreover all foreign keys and unique constraints have to be redefined. Therefore this results in a manual statement like the following:

```
CREATE TABLE IF NOT EXISTS thread_kern(
   CHECK(category_pk = 2),
   PRIMARY KEY(pk),
   FOREIGN KEY(category_pk) REFERENCES category(pk),
   FOREIGN KEY(author_pk)   REFERENCES author(pk),
   UNIQUE(tid, mid)
) INHERITS (thread);';
```

As stated before, using a stored procedure the whole task of creating each individual category table can be automated, and therefore the procedure shown in Listing 5 can be used to create all the tables.

### Step 3: Migrating Existing Data to the Right Table

Having all the per-category tables in place it is possible to migrate each tuple from the main *thread* table

**Listing 11.** *The first messages on the thread table before migrating data*

```
forumdb=> SELECT * FROM thread WHERE tid = 1;
 pk  | tid | mid |     title      |       content       | published_on | published_at | category_pk | author_pk
-----+-----+-----+----------------+---------------------+--------------+--------------+-------------+-----------
   1 |   1 |   0 | Thread start 1 | How do you do this? | 1991-01-11   | 08:02:00     |           5 |         1
   2 |   1 |   1 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:04:00     |           5 |         2
   3 |   1 |   2 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:06:00     |           5 |         3
   4 |   1 |   3 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:08:00     |           5 |         4
…
```

**Listing 12.** *The first messages are now on the hardware category table*

```
forumdb=> SELECT * FROM thread_hw WHERE tid = 1;
 pk  | tid | mid |     title      |       content       | published_on | published_at | category_pk | author_pk
-----+-----+-----+----------------+---------------------+--------------+--------------+-------------+-----------
   1 |   1 |   0 | Thread start 1 | How do you do this? | 1991-01-11   | 08:02:00     |           5 |         1
   2 |   1 |   1 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:04:00     |           5 |         2
   3 |   1 |   2 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:06:00     |           5 |         3
   4 |   1 |   3 | Thread reply 1 | I think so and so.  | 1991-01-11   | 08:08:00     |           5 |         4

...

forumdb=> SELECT * FROM ONLY thread WHERE tid = 1;
 pk | tid | mid | title | content | published_on | published_at | category_pk | author_pk
----+-----+-----+-------+---------+--------------+--------------+-------------+-----------
(0 rows)
```

**Listing 13.** *The stored procedure that creates all the tables for categories and years within a category*

```
CREATE OR REPLACE FUNCTION create_category_tables()
RETURNS integer
AS
$BODY$
DECLARE
        current_category        category%rowtype;
        created_tables          integer;
        current_table_name      text;
        current_year            integer;

        current_year_to_check   integer;
        current_max_year        integer;
BEGIN

        created_tables := 0;
        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP



                -- build a dynamic query for creating the
                    table
                --EXECUTE 'DROP TABLE thread_' ||
                    current_category.id;
                EXECUTE 'CREATE TABLE IF NOT EXISTS
                    thread_' || current_category.id
                        || ' ( CHECK(category_pk = ' ||
                    current_category.pk || '), '
                        || ' PRIMARY KEY(pk), '
                        || ' FOREIGN KEY(category_pk)
                    REFERENCES category(pk), '
                        || ' FOREIGN KEY(author_pk)
                    REFERENCES author(pk), '
                        || ' UNIQUE(tid, mid) '
                        || ') INHERITS (thread);';

                created_tables := created_tables + 1;

                -- compute the current year
                current_year := EXTRACT(year FROM
                    current_category.since);
                -- get the max year for the current
                    category, so that
                -- no more tables than the max year will
                    be created
                SELECT EXTRACT( year FROM MAX( published_on ) )
                  INTO   current_max_year
                  FROM   thread
                 WHERE  category_pk = current_category.pk;

                 RAISE LOG 'Generating time tables from
                     year % to year %', current_year,
                     current_max_year;

                WHILE current_year <= current_max_year  LOOP
                       RAISE LOG 'Creating sub-table for
                     year %', current_year;
                       current_year_to_check := EXTRACT(
                     year FROM current_category.since ) +
                     current_year - 1;


                EXECUTE 'CREATE TABLE IF NOT EXISTS thread_'
                         || current_category.id || '_
                     year' || current_year
                         || ' ( '
                         || '   CHECK( '
                         || ' EXTRACT(year FROM
                     published_on)  = '
                         || current_year
                         || ') ,'
                         || ' PRIMARY KEY(pk), '
                         || ' FOREIGN KEY(category_pk)
                     REFERENCES category(pk), '

                         || ' UNIQUE(tid, mid) '
                         || ') INHERITS ('
                        || 'thread_' || current_category.id
                         || ');';



                            current_year := current_year + 1;

                 END LOOP;  -- end of the per-year-while

        END LOOP;       -- end of the category iteration

        RETURN created_tables;

END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 14.** *The stored procedure to migrate existing partitions and split them depending on the time*

```
CREATE OR REPLACE FUNCTION migrate_threads_by_category_
                    and_time()
RETURNS VOID
AS
$BODY$
DECLARE
        current_category        category%rowtype;
        current_year            integer;
        current_year_to_check   integer;
        current_max_year        integer;
BEGIN

        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP


            current_year := EXTRACT(year FROM current_
                    category.since);

            --find the max year for this category
            SELECT EXTRACT( year FROM MAX( published_on
                    ) )
            INTO   current_max_year
            FROM   thread
            WHERE  category_pk = current_category.pk;


            WHILE current_year <= current_max_year LOOP


                        current_category.
                    description, current_year;
                -- copy each tuple in the right table
                EXECUTE 'INSERT INTO thread_' ||
                    current_category.id
                         || '_year' || current_year
                         || ' SELECT * FROM ONLY
                    thread_' || current_category.id
                         || ' WHERE category_pk = ' ||
                    current_category.pk
                         || ' AND ( EXTRACT( year FROM
                    published_on ) '
                         || ' = ' || current_year
                         || ')';

                -- delete tuples from the master table
                RAISE INFO 'Deleting for category %,
                    year %',
                             current_category.
                    description, current_year;
                EXECUTE 'DELETE FROM ONLY thread_' ||
                    current_category.id
                         || ' WHERE category_pk = ' ||
                    current_category.pk
                         || ' AND ( EXTRACT( year FROM
                    published_on ) '
                         || ' = ' || current_year
                         || ')';

                current_year := current_year + 1;

              END LOOP;      -- end of the per-year loop

            END LOOP;       -- end of the category iteration

END;
$BODY$
LANGUAGE plpgsql;
```

**Listing 15.** *An extract of the result of data partitioning based on categories and times.*

```
forumdb=> SELECT relname, reltuples FROM pg_class WHERE
                    relname like 'thread%' AND relkind =
                    'r' ORDER BY relname;
       relname        | reltuples
----------------------+-----------
 thread               |         0
 thread_hw            |         0
 thread_hw_year1991   |     29014
 thread_hw_year1992   |     29829
 thread_hw_year1993   |     29666
 ...
 thread_hw_year2004   |      5053
 thread_kern          |         0
 thread_kern_year1993 |     43621
 thread_kern_year1994 |     43862
 thread_kern_year1995 |     44103
 thread_kern_year1996 |     44103
 thread_kern_year1997 |     43862
 ...
 thread_kern_year2012 |     13255
```

to the per-category table to which it belongs: each tuple must be first inserted in the right table and then deleted from the *main* table. For instance, to migrate the "net" category tuples a transaction should issue the following:

```
INSERT INTO thread_net SELECT * FROM thread WHERE
                    category_pk = 2;
DELETE FROM ONLY thread WHERE category_pk = 2;
```

And similarly, this should be done for all other categories, as shown by the function *migrate_threads* in Listing 6. Careful readers will have noticed a new keyword, *ONLY*, in the DELETE query above. Almost every PostgreSQL command is aware of the table inheritance and accepts an *ONLY* clause to specify that the command must be routed exactly to the table without having to consider the inheritance chain. Therefore, in the above commands, the INSERT is performed in a child table, while the DELETE only on the parent table.

Before the migration the situation on the thread table was the following:

```
forumdb=> SELECT category, count(message)
FROM vw_thread
GROUP BY category;

  category  |  count
------------+---------
 Networking | 1493115
 Kernel     |  848802
 Hardware   |  391200
 General    | 1310540
```

while after the migration the situation is the same, with regard to the number of threads per category, but has a different layout (each category is now contained in a specific table and the *thread* table is empty):

```
forumdb=> SELECT relname, reltuples::integer
FROM pg_class WHERE relname LIKE 'thread%' AND relkind = 'r';
   relname   | reltuples
-------------+-----------
 thread      |         0
 thread_hw   |    391200
 thread_kern |    848802
 thread_misc |   1310540
 thread_net  |   1493115
```

Please note that for the above query to report the right result it might be required to run a VACUUM FULL ANALYZE.

**Step 4:**
**Avoiding the Manipulation of the Thread Table**

This step can be implemented either with rules or triggers on the *thread* main table: the idea is to intercept a statement routed to the thread table (e.g., INSERT) and re-route it to the appropriate table depending on the category information within the statement itself. For instance, when a new message is posted within the *net* category (*pk = 2*), the tuple must be stored in the *thread_net* table and not in the *thread* table.

**Using Rules**

Rules allow the rewriting of queries so that the real target (i.e., the table against which the statement will be executed) can be altered as required. For instance, for an INSERT statement to be routed to the "net" table (*pk = 2*) a rule like the following is required: Listing 7.

Such a rule, "attached" to the *thread* table, will re-route an insert of the "net" category into the *thread_net* table. Again, in order to get the rule creation automated, a stored procedure is used (see Listing 8). It is worth reminding readers that UPDATE, SELECT and DELETE statements already include the children tables, and therefore there is no explicit need to define rules for such queries, even if this were possible.

It is worth noting that the creation of rules, in particular the delete ones, will avoid manipulation of the *thread* table. This is the reason why data was migrated in the previous step, otherwise it would have been impossible to remove scattered tuples from the *thread* table. By the way, it is always possible to disable and re-enable the delete tuples in order to migrate data after this step.

**Using Triggers**

Since INSERT statements need re-routing to the right table, and this has to be applied before the tuple commits to the *thread* main table, it is possible to build a simple trigger function (see Listing 9) that can be associated with the *thread* table.

The *thread_partitioning_handler* trigger function acts for "before" events and performs an INSERT into the right table. It is worth noting the usage of the special syntax *EXECUTE...USING* that allows for the usage of the *NEW* trigger tuple. Moreover, please note that the trigger always returns *NULL* meaning that no tuple must hit the *thread* table as result of an INSERT statement. The trigger function is attached to the *thread* table via a stored procedure (see Listing 10), so that each time the procedure Is called the triggers for each available category are generated.

**Listing 16.** *The stored and trigger procedures to enable INSERT triggers on per-category tables*

```
CREATE OR REPLACE FUNCTION create_category_time_
                  triggers()
RETURNS void
AS
$BODY$
DECLARE
        current_category        category%rowtype;
BEGIN

        -- iterate over each category
        FOR current_category IN SELECT *
                                FROM category
                                ORDER BY id
                                LOOP


        EXECUTE 'DROP TRIGGER IF EXISTS tr_thread_
                    trigger_time ON '
                || ' thread_' || current_category.id;

        EXECUTE 'CREATE TRIGGER tr_thread_trigger_time '
                || ' BEFORE INSERT '
                || ' ON thread_' || current_category.id
                || ' FOR EACH ROW '
                || ' EXECUTE PROCEDURE thread_
                    subpartitioning_handler()';

        END LOOP;

END;
$BODY$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION thread_subpartitioning_
                  handler()
RETURNS trigger
AS
$BODY$
DECLARE
        current_category_id             text;
        current_category_table_name     text;
        current_category_since          date;
        current_category_year           integer;
BEGIN

        RAISE LOG 'Trigger executing as %', TG_OP;

        -- get the name of the category
        IF TG_OP = 'INSERT' THEN

                -- get the current category id and
                    current year
                SELECT id, EXTRACT( year FROM NEW.
                    published_on )
                FROM   category
                INTO   current_category_id, current_
                    category_year


                -- build the table name for the right
                    insert
                SELECT 'thread_' || current_category_id
                    || '_year'
                                || current_category_
                    year
                INTO   current_category_table_name;


        END IF;


        RAISE LOG 'The query is going to be re-routed to
                    the table %', current_category_table_
                    name;

        -- execute the insertion into the right table
        IF TG_OP = 'INSERT' THEN

                EXECUTE 'INSERT INTO '
                        || current_category_table_name
                        || ' SELECT $1.* '
                        USING NEW;
        END IF;

        RAISE LOG 'Trigger finished!';
        RETURN NULL;

END;
$BODY$
LANGUAGE plpgsql;
```

### Step 5: Provide Transparent Querying of the Thread Table

Believe it or not, this is the easiest part since PostgreSQL gives you this for free! Each `SELECT` targeted to the *thread* table will be automatically perform an union of all the results of the *thread* children: Listing 11. Which reports the same results as if the query were run against the single *thread_hw* table (*category_pk = 5*): Listing 12.

This is because the `SELECT` statement automatically provides a union of the table against which it is run (e.g., *thread*) and all its children. Using the ONLY keyword it is possible to tell PostgreSQL to avoid descending the inheritance chain, so that the following query now fails since the *thread* table is empty: Listing 12. Similar considerations are true for other non-insert statements, so there is no special need to build a specific set of rules/triggers.

## Time-based Partitioning

It is possible to improve the partitioning just completed by adding a new partition based on the thread time (i.e., the *published_on* column). The idea is to refine each per-category table by setting up a table that will contain each year of messages. This means that per-year tables will be defined for each category, for instance *thread_net_year1992*, *thread_net_year1993* and so on up to the last year a post for the category exists; each post will be then be routed to the right year table.

The partitioning steps are similar to those explained in the previous example, but now the first step is to create per-year tables, and therefore the stored procedure *create_category_tables* is changed as shown in Listing 13. After the tables are in place, data must be migrated from per-category tables to per-year tables, and this is done similarly to previous partitioning using the *migrate_threads_by_category_and_time* stored procedure (see Listing 14), that produces a database populated as shown in Listing 15. As a last step, a trigger to handle `INSERT` statements over each per-category table has to be set, and this is automated through the *create_category_time_triggers* stored procedure shown in Listing 16.

In the final scenario the master table *thread* cascades queries to per-category tables (e.g., *thread_net*) which in turn cascade to per-year tables within the same category (e.g., *thread_net_year2004*).

As readers can see, the above examples are quite simple and partially simulate a real situation. Of course having the ability to pre-partition the database before having to deal with huge amounts of data allows a better database design, and nullifies the migration of data. For instance, having chosen from the beginning to partition messages by time instead of by category could have remove a layer of inheritance (the per-category tables) that could result in a more difficult to maintain database.

## Summary and Coming Next

This article examined the data partitioning and the table inheritance that can be exploited in PostgreSQL to achieve data splitting across multiple tables. In the next article the tablespace feature will be presented as wells as another data partitioning scenario.

### Box 1. How to quickly (re)set the database

In order to perform the partitioning examples described here it is possible to quickly drop and rebuild the whole database. The source code of the example is contained in the GitHub repository (see the references) in the bsdmag/05-partitioning folder, therefore, being connected to the forumdb it is possible to issue the following queries:

```
DROP TABLE IF EXISTS thread_net CASCADE;
DROP TABLE IF EXISTS thread_misc CASCADE;
DROP TABLE IF EXISTS thread_kern CASCADE;
DROP TABLE IF EXISTS thread_misc CASCADE;
DROP TABLE IF EXISTS thread CASCADE;
DROP TABLE IF EXISTS author;
DROP TABLE IF EXISTS category;
DROP VIEW IF EXISTS vw_thread;
\i 01-forum-database-initial-setup.sql
\i 02-function-populate.sql
SELECT populate_forum();
\i 03-function-create-category-tables.sql
SELECT create_category_tables();
\i 04-migration.sql
SELECT migrate_threads();
\i 05-thread-table-rules.sql
SELECT create_category_rules();
```

And to enable the per-year subpartitioning:

```
\i 07-partitioning-time.sql
SELECT migrate_threads_by_category_and_time();
SELECT create_category_time_triggers();
and last issue
VACUUM FULL ANALYZE;
```

### On The Web

- PostgreSQL official Web Site: *http://www.postgresql.org*
- ITPUG official Web Site: *http://www.itpug.org*
- PostrgeSQL Table Inheritance Documentation: *http://www.postgresql.org/docs/current/static/ddl-inherit.html*
- GitHub Repository containing the source code of the examples: *https://github.com/fluca1978/fluca-pg-utils*

**LUCA FERRARI**

***Luca Ferrari lives in Italy with his wife and son. He is an Adjunct Professor at Nipissing University, Canada, a co-founder and the vice-president of the Italian PostgreSQL Users' Group (ITPUG). He simply loves the Open Source culture and refuses to log-in to non-Unix systems. He can be reached on line at http://fluca1978.blogspot.com.***

# Great Specials

## On FreeBSD & PC-BSD Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
**1.925.240.6652**

**$39.95**
FreeBSD 9.0 Jewel Case CD Set
or FreeBSD 9.0 DVD

**$29.95**
PC-BSD 9.0 DVD

**$49.95** Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.0 DVD

**$99.95**
The FreeBSD CD or DVD Bundle

**Inside each CD/DVD Bundle, you'll find:**
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.0 CD or DVD set
FreeBSD Toolkit DVD

*Stylish Dress Attire*
Look Your Professional Best


*Comfy Hoodies*
Stay Warm in Pullovers & Zip Ups

*T-Shirts*
Lots of Styles to Choose From

### FreeBSD 9.0 Jewel Case CD/DVD ..... $39.95

CD Set Contains:

- **Disc 1:** Installation Boot LiveCD (i386)
- **Disc 2:** Essential Packages Xorg, GNOME2 (i386)
- **Disc 3:** Installation Boot LiveCD (amd64)
- **Disc 4:** Essential Packages Xorg, GNOME2 (amd64)

FreeBSD 8.2 CD ..... $39.95
FreeBSD 8.2 DVD ..... $39.95

### FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.0 ..... $29.95
FreeBSD Subscription, start with DVD 9.0 ..... $29.95
FreeBSD Subscription, start with CD 8.2 ..... $29.95
FreeBSD Subscription, start with DVD 8.2 ..... $29.95

### PC-BSD 9.0 DVD (Isotope Edition)

PC-BSD 9.0 DVD ..... $29.95
PC-BSD Subscription ..... $19.95

### The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) ..... $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) ..... $39.95

### The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) ..... $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.0 ..... $79.95

### PC-BSD 9.0 Users Handbook ..... $24.95

### BSD Magazine ..... $11.99

### The FreeBSD Toolkit DVD ..... $39.95

### FreeBSD Mousepad ..... $10.00

### FreeBSD & PCBSD Caps ..... $20.00

### BSD Daemon Horns ..... $2.00


*Bundle Specials!*
Save $$$

*Just Plain Fun*
Mousepads & Novelty Horns



*BSD Magazine*
Available Monthly


For even **MORE** items
visit our website today!

**www.FreeBSDMall.com**

# Securing DNS Transactions

In the June 2012 issue, we outlined the threats, security objectives, and protection approaches for various DNS transactions.

### What you will learn…

- How to restrict transactions based on IP address
- How to configure TSIG for BIND and NSD

### What you should know…

- An understanding of how DNS works

This article provides the steps involved in implementing those approaches, as well as operational best practices that go with those implementations. The DNS protocol protection approaches are elaborated here in more detail:

**Restricting Transaction Entities Based on IP Address**
In this type of implementation, the DNS name servers and clients participating in a DNS transaction are restricted to a trusted set of hosts by specifying their IP addresses in appropriate access control statements provided by the name server software. The protection provided by these IP-based access control statements can be circumvented by attacks such as IP spoofing. Hence, this solution is not recommended for DNS query/response, zone transfer, and dynamic update transactions that have high threat impact. However, for the DNS NOTIFY transaction, where the only threat is spurious notification (which may not even trigger a zone transfer), an access control based on IP address will suffice. Although this solution is not recommended generally, a description of the mechanics of access control using IP addresses is provided in "Restricting Transaction Entities Based on IP Address" because the same statements are used to identify hosts based on named keys while implementing transaction protection using hash-based message authentication codes. This approach has been implemented for all DNS transactions.

**Transaction Protection through Hash-Based Message Authentication Codes (TSIG Specification)**
In this approach, transaction protection is enabled through generation and verification of *hash-based message authentication codes* (HMAC). Because these codes are embedded within a special RR of RRType TSIG, the specifications that outline protection of DNS transactions using HMAC are called TSIG in the DNS community. TSIG specifications are described in RFC 2845 and 3007. Application of TSIG specifications for protection of zone transfer and dynamic update transactions is described in *Transaction Protection Through Hash-Based Message Authentication Codes* (TSIG).

**Transaction Protection through Asymmetric Digital Signatures (DNSSEC Specification)**
This approach, which goes by the name DNS security extensions (DNSSEC), is described through a family of RFCs 4043, 4044, and 4045. The core services provided by DNSSEC are data origin authentication and integrity protection. DNSSEC is used mainly for securing DNS information obtained from DNS query/response transactions. The deployment issues of DNSSEC are described in Part 5.

Before we dive into the article, I want to take a moment to talk about NSD and Unbound vs. BIND. Most of you know that BIND is the *de facto standard* DNS server. It's

a free software product and is distributed with most UNIX and Linux platforms.

NSD is an open-source authoritative server developed by NLNet Labs of Amsterdam in cooperation with the RIPE NCC. NSD is a test-bed server for DNSSEC, and new DNSSEC protocol features are often prototyped using the NSD code base. The original intention of NSD was to develop an authoritative server implementation independent of BIND that could be used on root servers, thus making the root zone more robust through software diversity. Three root servers and several top-level domains now use NSD, but you don't have to be a root server or TLD to benefit from NSD's robustness, speed, and simplicity.

Unbound is a recursive DNS server that is complementary to NSD. It was developed in C by NLNet Labs from a Java implementation by Verisign, Nominet, Kirei, and EP.NET. Together, NSD and Unbound provide flexible, fast, secure DNS service appropriate for most sites. The NLNet Labs components are not as mature as BIND and do not have as many bells and whistles, but they are fine solutions for most sites.

The DNSSEC code in NSD and Unbound is more robust and better tested than that in BIND. It's also faster. For example, Unbound is about five times faster than BIND at verifying DNSSEC signatures. BIND still has an edge in some areas, though, notably in documentation and in extra features. For a really robust DNS regime, run both!

## Restricting Transaction Entities Based on IP Address

Some DNS name server implementations, such as BIND 9 and higher, provide access control statements through which it is possible to specify hosts that can participate in a given DNS transaction. The hosts can be identified by their IP address or IP subnet reference (called *IP prefix*) in these statements.

The list containing these IP addresses and/or IP prefixes is called an *address match list*. (An address match list can be made up of other things besides IP addresses and IP prefixes, as described in "Restricting DNS Query/Response Transaction Entities"). The address match list is used as an argument in various access control statements that are available for use in BIND configuration files. There are separate access control statements for each type of DNS transaction. The syntax of the various access control statements and the DNS transaction for which each is used are given in Table 1.

The purpose of each of these access control statements is as follows:

- *allow-query*: specifies the list of hosts allowed to query the name server as a whole or a particular zone within the name server
- *allow-recursion*: specifies the list of hosts allowed to submit recursive queries to the name server as a whole or to a particular zone served by the name server
- *allow-transfer*: specifies the list of hosts allowed to initiate zone transfer requests to the name server as a whole or to a particular zone within the name server. This statement is predominantly required for configuration of master name servers.
- *allow-update*: specifies the list of hosts allowed to initiate dynamic update requests
- *allow-update-forwarding*: specifies the list of hosts allowed to forward dynamic update requests (regardless of the originator of the requests)
- *allow-notify*: specifies the list of hosts from which to accept DNS NOTIFY messages indicating changes in the zone file. This list is relevant only for configuration of secondary slave name servers.
- *blackhole*: specifies the list of hosts that are blacklisted (barred) from initiating any transaction with this name server. Used only in an options server-wide ACL statement.

The foregoing access control statements are, in fact, substatements that can be used in the context of `options`

**Table 1.** *BIND Access Control Statement Syntax for DNS Transactions*

| Access Control Statement Syntax | DNS Transaction |
|---|---|
| allow-query { address_match_list } | DNS Query/Response |
| allow-recursion { address_match_list } | Recursive Query |
| allow-transfer { address_match_list } | Zone Transfer |
| allow-update { address_match_list } | Dynamic Update |
| allow-update-forwarding { address_match_list } | Dynamic Update |
| allow-notify { address_match_list } | DNS Notify |
| blackhole { address_match_list } | Blacklisted Hosts |

and `zone` statements in the BIND 9 and higher configuration file (with the exception of *blackhole*). When they are used within the `zone` statement, they specify access control restrictions for the corresponding DNS transaction for that specific zone. When they are used as part of the `options` statement, they specify access control restrictions for the corresponding DNS transaction for the name server as a whole (because a name server could host multiple zones).

NSD has a similar set of configuration options for certain transactions. NSD has a more limited set of options and currently can only restrict zone transfers. In the following sections, if a comparable set of options exists for NSD, they will be listed. Otherwise, it should be noted that a comparable option does not exist at the time of writing.

## Restricting DNS Query/Response Transaction Entities

An example of the usage of the `allow-query` substatement (to specify restrictions for the DNS query/response transaction stating the IP addresses/subnets from which DNS queries are accepted) both at the server- and at the zone level (for the zone *example.com*) is given below:

```
options {
   allow-query { 254.10.20.10; 239.10.30.29/25; };
};

zone "example.com." {
   type master;
   file "zonedb.example.com";
   allow-query { 192.249.249.1; 192.249.249.4; };
};
```

Specifying the list of IP addresses and IP prefixes within the options and zone statements could clutter the configuration file. Furthermore, the list of IP addresses and IP prefixes could be the same for many of the access control statements within a name server, and errors could be introduced if any additions or subtractions are made for that list. To avoid these problems, BIND provides a means to create named address match lists, which are called *access control lists* (ACL). These ACLs can be used in place of the list of IP addresses/IP prefixes (in the address match list argument) in the access control statements.

The ACLs are created by using the `acl` statement in BIND 9 and higher. The general syntax of the `acl` statement is as follows:

```
acl acl-list-name {
   address_match_list
};
```

The `acl-list-name` is a user-defined string (e.g., `internal_hosts`). The `address_match_list` can be a list of IP addresses, IP address prefixes (denoting subnets), or cryptographic keys. An example of an `acl` statement that uses an IP address and a subnet reference in `address_match_list` is given below. In the example, 254.10.20.10 denotes the IP address of a host, and the IP prefix 239.10.30.0/24 denotes a class C subnet.

```
acl "internal_hosts" {
   254.10.20.10;
   239.10.30.0/24;
};
```

The use of ACL `internal_hosts` in place of the list of IP addresses/IP prefix in the `options` and `zone` statement given above is as follows:

```
options {
   allow-query { internal_hosts; };
};

zone "example.com." {
   type master;
   file "zonedb.example.com";
   allow-query { internal_hosts; };
};
```

The address match list parameter in an access control statement can contain any of the following values:

- An IP address or list of IP addresses
- An IP prefix or list of IP prefixes
- ACLs
- A combination of the above three.

The definition of ACLs forms a critical element in the configuration of DNS transaction restrictions. Hence, it is a good operational practice for the DNS administrator to define and create ACLs pertaining to different DNS transactions.

### Checklist #7

It is recommended that the administrator create a named list of trusted hosts (or blacklisted hosts) for each of the different types of DNS transactions. In general, the role of

the following categories of hosts should be considered for inclusion in the appropriate ACL: (1) DMZ hosts defined in any of the zones in the enterprise; (2) All secondary name servers allowed to initiate zone transfers; (3) Internal hosts allowed to perform recursive queries.

In addition to IP address, IP prefix, or ACL, the address match list parameter in the access control statements can take on any of the following special values:

- *none*: matches no hosts
- *any*: matches all hosts
- *localhost*: matches all IP addresses of the server on which the name server is running
- *localnets*: matches all IP addresses and subnet masks of the server on which the name server is running.

Following are a few more examples of commands for creating ACLs and the use of ACLs within `options` and `zone` statements:

```
acl "local_hosts" {
254.10.20.10;
239.10.30.29/25;
};

acl "fake-net" {
0.0.0.0/8;
1.0.0.0/8;
};

options {
   allow-query { any; };
   blackhole { fake-net; };
};

zone "example.com." {
   type master;
   file "zonedb.example.com";
   allow-query { local_hosts; };
};
```

In the `named.conf` snippet above, two ACLs, `local_hosts` and `fake-net`, have been specified. DNS queries from any hosts are allowed at the server level. No transactions are permitted from the hosts included under `fake-net`. Queries to the zone *example.com* can be initiated only by the hosts included under the ACL `local_hosts` because any restriction specified under the `zone` (zone-specific) statement overrides the restriction specified under the `options` (server-wide) statement.

Key material can also be used in ACL statements. This would indicate that only hosts knowing (and using) the shared key (or key pair) would be able to communicate. How a key is used in an ACL is discussed in "Defining the Keys in the Communicating Name Servers."

NSD does not have a feature to define ACLs as a means of only allowing queries from designated sets of hosts.

## Restricting Recursive Queries

Authoritative name servers provide name resolution service from their own data and are supposed to provide this service for any DNS client. Hence, configuring an authoritative name server to accept queries from a restricted set of hosts does not make sense. The practical security protection for an authoritative name server is to turn off the query recursion feature, so that the authoritative name server does not poison its cache by querying other (possibly compromised) name servers. A local resolving/recursive name server can be configured to accept queries only from internal hosts, to protect it from denial-of-service attacks as well as cache poisoning. However, there may be situations in which it is economically infeasible to dedicate separate servers for authoritative service and resolution service, and the resolving name server has to perform as authoritative server for one or more zones. In this situation, the following strategies are possible within the BIND 9 and higher name server:

- Restricting all queries accepted by the server to a specified set of IP addresses of internal clients and then overriding this set only for authoritative zones so that any DNS client can obtain information for resources in that zone.
- Restricting recursive queries to a specified set of IP addresses of internal clients through a direct configuration option
- Serving different responses (data) to different clients by defining views.

### Restriction at server level with override for authoritative zones

In this strategy, the allowable set of internal clients who can submit queries to the name server is specified through the acl statement as follows:

```
acl internal_hosts {192.158.43.3; 192.158.43.6;
                    192.158.44.56;};
```

The server-wide option would be to restrict all queries to the list of clients:

```
options {
   allow-query { internal_hosts; };
   - or -
   allow-recursion { internal_hosts; };
};
```

The option can be overridden by specifying zones for which this name server is authoritative (thus allowing queries to that zone from all clients):

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   allow-query { any; };
};
```

**Restricting all recursive queries to a specified set of IP addresses**

Server-wide restriction:

```
options {
   allow-recursion { internal_hosts; };
};
```

**Restricting recursion through views**

The purpose of creating views is to create a logical partition made up of a combination of clients (based on IP addresses) and zones for which recursive queries will be supported and those for which they will not be supported.

In the following example, the view `recursion_view` is enabled to define the scope of IP addresses and zones that are permitted to submit recursive queries; `no_recursion_view` is meant for disallowing recursion.

```
view recursion_view {
   match-clients { internal_hosts; };
   recursion yes;
};

view no_recursion_view {
   match-clients { any; };
   recursion no;
};
```

It should be noted that NSD (as of the time of writing) is an authoritative only DNS server.

Therefore, an NSD server will never act as a recursive server and only serve authoritative information from the zones it is configured to serve.

## Restricting Zone Transfer Transaction Entities

Authoritative name servers (especially primary name servers) should be configured with an `allow-transfer` access control substatement designating the list of hosts from which zone transfer requests can be accepted. These restrictions address the denial-of-service threat and potential exploits from unrestricted dissemination of information about internal resources.

Based on the need-to-know principle, the only name servers that need to refresh their zone files periodically are the secondary name servers. Hence, zone transfers from primary name servers should be restricted to secondary name servers. The zone transfer should be completely disabled in the secondary name servers. The address match list argument for the `allow-transfer` substatement should consist of IP addresses of secondary name servers and stealth secondary name servers.

The command to create an ACL `valid_secondary_NS` with the IP addresses of three secondary name servers is as follows:

```
acl "valid_secondary_NS" {
   224.10.229.5;
   224.10.235.6;
   239.10.245.25;
};
```

The `allow-transfer` substatement can be used in a `zone` statement and in an `options` statement. When it is used in a `zone` statement, it can restrict zone transfer for that zone; when it is used in an `options` statement, it can restrict zone transfer for all zones in the name server.

The `allow-transfer` substatement at the server level is as follows:

```
options {
   allow-transfer { "valid_secondary_NS"; };
};
```

The `allow-transfer` substatement at the zone level is as follows:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   allow-transfer { "valid_secondary_NS"; };
};
```

The foregoing statements apply to primary name servers. In the secondary and stealth secondary name servers, zone transfer should be disabled as shown below:

```
zone "example.com" {
   type slave;
   masters { 224.239.5.1; };
   file "zonedb_bak.example.com";
   allow-transfer { none; };
};
```

## Restricting Zone Transfer in NSD

NSD has a similar set of tools to restrict zone transfers to only a chosen set of slave servers. Like BIND, the administrator should learn and use the options available in the NSD configuration file. There is no way to create access control lists (ACLs), but an administrator can list the individual IP addresses of slave servers in the zone statements in the NSD configuration file.

In the configuration file, the `provide-xfr` statement is used in the zone statement block of the `nsd.conf` file, much like a combined `masters` statement and `allow-transfer` statement in BIND configuration files:

```
zone:
   #allow transfer from subnet
   provide-xfr: 169.192.85.0/24
   #prevent transfer from specific IP address in block
   provide-xfr: 169.192.85.66 BLOCKED
```

Only one IP address should appear in a `provide-xfr` statement, but the address can be an entire subnet. The `provide-xfr` statement allows transfers; all other transfer requests are rejected by default.

## Restricting Dynamic Update Transaction Entities

Dynamic updates on a zone file can be directed only to the copy of the zone file that resides on the primary name server for the zone (i.e., where the master zone file resides). By default, dynamic update is turned off in both BIND 8 and higher. Dynamic updates are enabled or restricted by using one of the following two statements in BIND:

- allow-update
- update-policy (available only in BIND 9 and higher).

These statements can be specified only at the zone level, not at the server level. Hence, these statements are substatements within the `zone` statement. The `allow-update` substatement enables specification of dynamic update restrictions based on IP addresses and a shared secret (also called a *TSIG key*[1]). The use of the `allow-update` statement using IP addresses alone is addressed in this section. The use of the `allow-update` statement using TSIG keys is described in "Securing Zone Transfers using TSIG."

The `update-policy` statement enables specification of dynamic update restrictions based on TSIG keys only, but it enables specification of update restrictions at a finer level of granularity. The `allow-update` substatement implies update access rights to all records of a zone; the `update-policy` substatement can be used to restrict update access rights to one or more designated RRTypes (e.g., A RRs).

To use the `allow-update` statement, an address match list must be created. The command to create an ACL `DU_Allowed_List` with one IP address is as follows:

```
acl "DU_Allowed_List" {
   192.249.12.21;
};
```

The ACL `DU_Allowed_List` (consisting of IP addresses of hosts allowed to send dynamic update requests for updating the contents of the zone `example.com`) is used within the `allow-update` substatement of the `zone` statement as follows:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   allow-update { "DU_Allowed_List"; };
};
```

Dynamic update requests generally originate from hosts such as DHCP servers that assign IP addresses dynamically to hosts. Once they assign an IP address to a new host, they need to store the FQDN-to-IP address mapping (by creating an A RR) and address-to-FQDN mapping (by creating a PTR RR) information in the primary authoritative name servers for the zones. Creation of this information occurs through dynamic updates.

As of the time of writing, NSD does not support dynamic update so there are no comparable configuration options for NSD. All dynamic update messages sent to a DNS server running NSD will be rejected. Updates to a zone must be done offline and then then the server signaled to reload the new modified zone.

## Restricting BIND DNS NOTIFY Transaction Entities

Once zone transfers have been set up between servers, it is a good idea to make sure that secondary name servers are informed about changes to zone file data through a notification message. By default, a notification message is sent whenever a primary name server detects a change in the zone file. It sends a DNS NOTIFY message to every name server listed in the NS RRSet in the zone, because they are the recognized secondary name servers of the zone. DNS administrators should keep notification on, as this configuration will allow updates to be propagated quickly to secondary name servers. If the DNS administrator wants to turn off the functionality for a specific zone, however, the `notify` substatement should be used in the `zone` statement of that zone:

```
zone "example.com" {
   type master;
   notify no;
   file "zonedb.example.com";
};
```

If there are any additional servers to which the zone administrator wants the DNS NOTIFY message to be sent (e.g., a stealth slave server), the `also-notify` substatement should be added to the `zone` statement, and the IP addresses of the additional servers should be specified as its parameter values, as shown below:

```
zone "example.com" {
   type master;
   also-notify { 192.168.25.2; };
   file "zonedb.example.com";
};
```

The receiver of the DNS NOTIFY message, the secondary name server, allows notify messages only from the primary name server by default. (Recall that the secondary name server is made aware of its primary name server through the `masters` substatement in the `zone` statement.) If the secondary name server wants to receive notify messages from additional servers, the `allow-notify` substatement in the `zone` statement must be added, and then the IP addresses of those servers must be specified in that substatement, as follows:

```
zone "example.com" {
   type slave;
   allow-notify { 193.168.25.4; };
   file "zonebak.example.com";
   masters { 192.168.25.1; };
};
```

## Restricting NSD DNS NOTIFY Transaction Entities

There are two statements (both placed in the `zone:` statement block) that a DNS administrator can use to send DNS NOTIFY messages or restrict listening for DNS NOTIFY messages to a particular IP address (a master server in the case of NSD acting as a slave server).

To configure NSD to send DNS NOTIFY messages to a particular IP address (either a slave secondary or a stealth secondary) and a particular TSIG key or the option `NOKEY` if no TSIG is used, the following is added to the `zone:` statement block in the NSD configuration file:

```
zone:
   notify: 10.0.0.10 NOKEY
```

The configuration of a slave server, accepting notification messages only from specific IP addresses, would look like the following:

```
zone:
   allow-notify: 10.11.12.13 NOKEY
```

## Transaction Protection through Hash-Based Message Authentication Codes (TSIG)

The process of authenticating the source of a message and its integrity through hash-based message authentication codes (HMAC) is specified through a set of DNS specifications known collectively as TSIG. The term HMAC is used to denote both the message authentication code generated by using a keyed hash function and the hash function itself. HMAC is specified in RFC 2104 and generalized in the NIST document FIPS 198-1.

An HMAC function uses two parameters – a message input and a secret key – and produces an output called a *message authentication code* (MAC) or *hash.* The sender of the message uses the HMAC function to generate a MAC and sends this MAC along with the message to the receiver. The receiver, who shares the same secret key, uses the key and HMAC function used by the sender to compute the MAC on the received message. The receiver then compares the computed MAC with the received MAC; if the two values match, it provides assurance that the message has been received correctly and that the sender belongs to the community of users sharing the same secret key. Thus, message

source authentication and integrity verification are performed in a single process.

The hash algorithm, which forms the primitive for the hash function, generates a fixed-size MAC or hash from a message of arbitrary size. The HMAC function for TSIG is specified in RFC 2845 and was extended in RFC 4635 to support more hash algorithms (SHA-1 and SHA-2 family of algorithms).

Transaction protection through HMAC, using a shared secret, is not a scalable solution. This is the reason the TSIG specification is largely used only for zone transfer and dynamic update transactions. These DNS transactions are either between servers in the same administrative domain, or between servers in domains with previously established interactions interconnections.

The MAC, or hash value generated by the sender of the DNS message, is placed in a new RR called a TSIG record that is added to the DNS message. The TSIG record, in addition to the generated hash, contains the following:

- Name of the hash algorithm used
- Key name
- Time the hash was generated (time stamp)
- "Fudge factor" – time in seconds (usually 5 minutes) to use as delta on either side of the time generated for which the TSIG signature should be considered valid; used to account for possible clock skew between hosts.

The time stamp field specifies the time at which the MAC was generated. The purpose of this field is to protect against replay attacks. In a replay attack, the attacker could capture the packet containing the MAC, and send it after a period of time. To ensure that this does not happen, the recipient reads the MAC generation time and the current clock time, and verifies whether the MAC was generated within an "allowable expiry time," which is computed using the "Fudge Factor".

The "Fudge factor" field specifies the duration of time after the MAC generation time, the message can be considered valid. It is computed by applying a "fudge factor" on the MAC generation time (adding or subtracting a small number of seconds) to allow for clock skew (mismatch) between the MAC generator and verifier hosts.

To have a secure transaction based on TSIG, a sender computes the hash of the entire DNS message and secret key, and encodes the result in a TSIG RR at the end of the message. At the recipient end, the TSIG record is stripped from the DNS message and processed. The process whereby the recipient uses the TSIG record to verify the integrity of the received DNS message is called verification. The verification process uses the hash algorithm name to identify the hash function, and the key name to identify the key to be used to validate the TSIG record. The number of the fudge factor is used to add to, or subtract from the signing time, to allow for the possible mismatch of clocks of the signer and verifier. Thus, the fudge factor provides the tolerance limit for the MAC validity period computed, based on the time of generation.

The purpose of sending the key name in the TSIG record is to enable the verifier (recipient) of the DNS message to use the right key to verify it. It also enables the recipient to verify that the key name is indeed one of the keys shared with the sender. The purpose of the "Time Signed" or time stamp field in the TSIG record is to inform the message recipient about the time of MAC generation. The recipient compares this value with the current clock time at the recipient system to ensure that the MAC was generated within the allowable time specified as part of the TSIG record itself. The purpose of using a time stamp is to prevent replay attacks. For correct verification of the generation time against the current time, it is essential that the system clocks of the transaction participants to be synchronized. Protocols such as the *Network Time Protocol* (NTP) are available for this purpose.

The verification process consists of the recipient retrieving the appropriate secret key, generating its own hash of the received DNS message, and comparing it with the received hash (in the TSIG record). In this verification process, the receiving name server has performed the following validations:

- The message has been verified as coming from an authenticated source (with whom it shares the secret key).
- The message has not been altered in transit (verified by matching hash values).

Source authentication counters identify spoofing, and data integrity checking helps to counter corruption and modification of data in transit.

BIND version 8.2 was the first version to introduce TSIG features, and is present in every later version. Support for TSIG in BIND 9 and higher includes features to secure zone transfer and dynamic update transactions [2].

The following operations are needed to set up the environment for enabling DNS transactions to use TSIG:

- The system clocks of the name servers (primary and secondary) participating in DNS transactions must be synchronized (e.g., through NTP).

- There should be a secret key generation utility that can generate keys of the required length with sufficient entropy. The key file, (the file containing the secret key string), must be securely communicated to the two servers participating in the transaction.
- The key information should be specified in the configuration file through appropriate statements (e.g., by the `key` statement and `server` statement in the `named.conf` configuration file of BIND 9 and higher).

The key generation process is described in "Key Generation." The commands needed to define the keys and instruct the name server to use those keys for all DNS transactions are outlined in "Defining the Keys in the Communicating Name Servers" and "Instructing Name Servers to Use Keys in All Transactions." The set of checklists for key file creation and key definition within the name servers is given in "Checklists for Key File Creation and Key Configuration Process." Protection of zone transfer transactions and dynamic update transactions using HMAC, as specified in TSIG, are covered in "Securing Dynamic Updates Using TSIG or SIG(0)" and "Configuring Dynamic Update Forwarding Restrictions Using TSIG Keys."

## Key Generation

To enable zone transfer (requests and responses) through authenticated messages, it is necessary to generate a key for every pair of name servers. The key can also be used for securing other transactions, such as dynamic updates, DNS queries, and responses.

The binary key string, that is generated by most key generation utilities used with DNSSEC, is base64 encoded. The program that generates the key in BIND 9 and higher is `dnssec-keygen`. An example of a command that generates a secret key (as opposed to other types of keys, such as public keys, which this program can also generate) by invoking the `dnssec-keygen` program is as follows:

```
dnssec-keygen -a HMAC-SHA256 -b 112 -n HOST ns1-ns2.
                    example.com.
```

where the various command options (parameters) denote the following:

- `-a` option: the name of the hashing algorithm that will use the key (HMAC-SHA256 is preferred, but may not be available in older implementations. Use of HMAC-SHA1 is allowed, but migration to HMAC-SHA256 should be done when available)
- `-b` option: the length of the key (here – 112 bits)
- `-n` option: the type of key (in this case, the `HOST`)
- last parameter: the name of the key (`ns1-ns2.example.com`)

The `dnssec-keygen` program generates the following files, each containing the key string:

```
Kns1-ns2.example.com.+157+34567.key
Kns1-ns2.example.com.+157.34567.private
```

When the program is generating a pair of keys (one public and the other private), the file with the extension *key* will contain the public key string and the file with extension private will contain the private key. Because in this case only the secret key is being generated, the key strings in both files will be the same for the TSIG implementation. The key string from any of these files is then copied to a file called the key file. This file is then referenced using an include statement within the `key` statement.

## Defining the Keys in the Communicating Name Servers

The key generated by using the `dnssec-keygen` utility has to be defined within the `named.conf` configuration file of the two communicating servers (generally one primary name server and one secondary name server). This is accomplished by using the `key` statement of BIND:

```
key "ns1-ns2.example.com." {
   algorithm hmac-sha256;
   include "/var/named/keys/secretkey.conf";
};
```

where the file `secretkey.conf` will contain the keyword secret and the actual key string (in this example):

```
secret "MhZQKc4TwAPkURM==";
```

## Defining the Keys in an NSD Configuration File

In NSD, declaring a TSIG key is very similar to the example above, with some minor syntax changes:

```
key:
   name: ns1-ns2.example.com.
   algorithm: hmac-sha256
   secret: "MhZQKc4TwAPkURM=="
```

## Instructing Name Servers to Use Keys in All Transactions

The command to instruct the server to use the key in all transactions (DNS query/response, zone transfer, dynamic update, etc.) is as follows:

```
server 192.249.249.1 {
   keys { ns1-ns2.example.com.; };
};
```

The same statement can be used as an entry in an `acl` statement as well:

```
acl key_acl {
   ns1-ns2.example.com.;
};
```

## Checklists for Key File Creation and Key Configuration Process

In each of the configuration files of the pair of servers that share a secret key in a zone, the name of the key to be used for all communication between them must be specified (via the `server` statement in BIND configuration files).

**Checklist #8**
The TSIG key should be a minimum of 112 bits in length, if the generator utility has been proven to generate sufficiently random strings. The generated TSIG key may have to be longer to insure at least 112 bits of security.

**Checklist #9**
A unique TSIG key should be generated for each pair of communicating hosts (i.e., a separate key for each secondary name server to authenticate transactions with the primary name server, etc.).

**Checklist #10**
After the key string is copied to the key file in the name server, the two files generated by the `dnssec-keygen` program should either be made accessible only to the server administrator account (e.g., root in UNIX) or, better still, deleted. The paper copy of these files also should be destroyed.

**Checklist #11**
The key file should be securely transmitted across the network to name servers that will be communicating with the name server that generated the key.

**Checklist #12**
The statement in the configuration file (usually found at `/etc/named.conf` for BIND running on UNIX) that describes a TSIG key (key name (ID), signing algorithm, and key string) should not directly contain the key string. When the key string is found in the configuration file, the risk of key compromise is increased in some environments where there is a need to make the configuration file readable by people other than the zone administrator. Instead, the key string should be defined in a separate key file and referenced through an include directive in the `key` statement of the configuration file. Every TSIG key should have a separate key file.

**Checklist #13**
The key file should be owned by the account under which the name server software is run. The permission bits should be set so that the key file can be read or modified only by the account that runs the name server software.

**Checklist #14**
The TSIG key used to sign messages between a pair of servers should be specified in the `server` statement of both transacting servers to point to each other. This is necessary to ensure that both the request message and the transaction message of a particular transaction are signed and hence secured.

## Securing Zone Transfers using TSIG

The pair of servers participating in zone transfer transactions must be instructed to use the key defined using the `key` statement (see "Defining the Keys in the Communicating Name Servers"). This pair generally consists of a primary name server and a secondary name server. The primary name server is configured to accept zone transfer requests only from secondary name servers that send MACs using the named key along with a zone transfer request message.

The configuration is accomplished by using the `allow-transfer` substatement of the `zone` statement. A sample `allow-transfer` substatement that specifies that the primary name server should only allow zone transfer requests for the `example.com` zone from name servers that use the `ns1-ns2.example.com` key is as follows:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   allow-transfer { key {ns1-ns2.example.com.}; };
};
```

The secondary name server is instructed to use the key `ns1-ns2.example.com` in the zone transfer request to the primary name server (with IP address 192.249.249.1) using the `server` statement shown in "Defining the Keys in an NSD Configuration File."

In NSD, the syntax is similar, but there is a special requirement if no TSIG key is to be used. The zone option `provide-xfer` is used to indicate which IP addresses can request a zone transfer for this zone on this server:

```
zone:
   type: master;
   file "zonedb.example.come";
   provide-xfer: 192.68.0.1 ns1-ns2.example.com.
   provide-xfer: 192.68.0.1 NOKEY
```

## Securing Dynamic Updates Using TSIG or SIG(0)

Dynamic update restrictions based on TSIG keys can be specified in BIND 8.2 and higher versions [3] by using the `allow-update` substatement of the `zone` statement. The arguments to this statement are the keyword `key` followed by the name of the TSIG key. (See "Defining the Keys in the Communicating Name Servers" for details on how to enter in the `key` statement in a BIND name server configuration file.) Once the `key` statement has been entered, the following substatement can be added to the `zone` statement to make use of the secret key for dynamic updates:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   allow-update { key dhcp-server.example.com.; };
};
```

Note that although the string `dhcp-server.example.com.` looks like a FQDN, it actually denotes the name of the TSIG key. The implication of the configuration statement example is that any hosts that possess the key named `dhcp-server.example.com.` can submit dynamic update requests (adding, deleting, or modifying RRs) to the zone file (for the zone `example.com`) that resides in the primary authoritative name server.

To use SIG(0) to authenticate dynamic update messages, the key used must first have its public component stored in the DNS, so a validating client can obtain it.[4] We'll discuss publishing public keys and setting up trust anchors in Part 5. The previous steps above need to be performed to control access (if desired). After that is done, the updating name server should be able to obtain the key and process the dynamic update request (if the name server supports SIG(0) with dynamic update).

## Configuring Dynamic Update Forwarding Restrictions Using TSIG Keys

Dynamic updates are allowed on the copy of the zone file in the primary authoritative name server only because that is the only "writable" copy. This does not automatically imply that the primary authoritative name server is the only one allowed to accept dynamic update requests.

In fact, BIND 9.1 and higher versions allow secondary name servers to accept dynamic update requests and forward them to the primary authoritative name server.

In this scenario, if there are no restrictions on the basis of the identity of hosts from whom the secondary name server can forward such dynamic update requests, it is equivalent to circumventing the dynamic update restrictions specified in the primary name server because the request can literally originate from any host to the secondary name server and be forwarded to the primary name server. To counter this problem, a new substatement, `allow-update-forwarding`, is now available in BIND versions that have the dynamic update forwarding feature. An example of this `allow-update-forwarding` statement using TSIG keys is given below:

```
zone "example.com" {
   type slave;
   file "backupdb.example.com";
   allow-update-forwarding { key dhcp-server.example.com.;
                  };
};
```

## Configuring Fine-Grained Dynamic Update Restrictions Using TSIG/SIG(0) Keys

The `allow-update` substatement specifies dynamic update restrictions based on the originators of dynamic update requests (a specific set of hosts identified by IP address or holding a TSIG key), but not the contents of the zone records. To specify dynamic update access (grant or deny) restrictions based on a combination of domain/subdomain names and RR types (A, MX, NS, etc.), BIND 9 and higher versions provide the `update-policy` substatement within the `zone` statement. The `update-policy` substatement bases these restrictions on the TSIG key. In other words, the `update-policy` statement specifies which TSIG keys (or holders of keys) are allowed to perform dynamic updates on which domains/subdomains and RR types within that domain/subdomain.

The general form of the `update-policy` statement is as follows:

```
update-policy {
   (grant | deny) TSIGkey nametype name [type]
};
```

where the semantics of each of the statement components are as follows:

- *grant/deny* – allow/disallow dynamic update for the combination that follows
- *TSIGkey* – the name of the TSIG key used to authenticate the update request
- *nametype* – can be one of the following with the associated semantics:
- *name* – restriction applies to the domain name specified in the following name field
- *subdomain* – restriction applies to subdomains of the domain specified in the following name field
- *wildcard* – restriction applies to the set of domains specified using the wildcard syntax (i.e., *) in the following name field
- *self* – restriction applies to the domain whose name is the same as that in the TSIG key field (i.e., the domain name whose records are to be updated has the same name as the key used to authenticate the dynamic update request). In this usage, the contents of the name field become redundant but still should be used in the statement (i.e., the name field cannot be left blank)
- *name* – used to specify the name of the domain. The syntax used and the domains it covers are based on the value used in the nametype field (e.g., if subdomain is the value of the nametype field, then all subdomains of the domain name used are being covered under this statement).
- *type* – an optional field that can contain any valid RRType (except the NSEC type) or the wildcard type 'ANY' (ANY stands for all RR types except the NSEC type). If it is missing, it denotes all RR types, except SOA, NS, RRSIG, and NSEC. It also is possible to put in multiple RRTypes separated by a space (e.g., A NS).

Examples of `update-policy` statements and their associated semantics are given below.

Suppose there is a domain `sales.example.com` within `example.com` and that name server uses a TSIG key that has the same name as its own domain name (i.e., `sales.example.com`). All dynamic updates from `sales.example.com` could be restricted to all resource records of that domain within the zone file as follows:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   update-policy { grant sales.example.com. self sales.
                   example.com.; };
};
```

All dynamic updates from `sales.example.com` could be restricted to only A and MX RR types of that domain as follows:

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   update-policy { grant sales.example.com. self sales.
                   example.com. A MX; };
};
```

To allow clients with the TSIG key `sales.example.com` to update all records pertaining to subdomains of `NEsales.example.com` except the name server records (RR Type NS):

```
zone "example.com" {
   type master;
   file "zonedb.example.com";
   update-policy {
      deny sales.example.com. subdomain NE sales.example.
                   com. NS;
      grant sales.example.com. subdomain NE sales.example.
                   com. ANY;
   };
};
```

For Microsoft Windows, authentication is provided using GSS-TSIG. System administrators of Windows Servers should consult their implementation's documentation on how to integrate secure dynamic update using GSS-TSIG.

## Summary

Transaction Signature (or TSIG) is a protocol defined in RFC 2845. It's used by DNS to provide a means of authenticating updates to a dynamic DNS database, although it can also be used between servers, and for regular queries. TSIG uses shared secret keys and one-way hashing to provide a cryptographically secure means of identifying each endpoint of a connection as being allowed to make, or respond to a DNS update.

Although queries to DNS may be made anonymously, updates to DNS must be authenticated, since they make lasting changes to the structure of the Internet naming system. The use of a key shared by the client making the update and the DNS server, guarantees the authentic-

## Footnotes

[1] The term TSIG Key (while commonly used) is not technically correct, as it refers to a shared secret string and not a cryptographic key.
[2] Some server software such as Microsoft Windows Server 2008 does not implement TSIG, but use lower level transaction security (such as IPSec). To set this up, see *http://technet.microsoft.com/en-us/library/ee649243(WS.10).aspx*
[3] As mentioned earlier, Micosoft Server does not use TSIG for dynamic updates, but uses IPSec instead. See: *http://technet.microsoft.com/en-us/library/cc753751.aspx*
[4] For more information, see the note from the dynamic update howto: *http://ops.ietf.org/dns/dynupd/secure-ddns-howto.html*
[5] *http://www.caida.org/outreach/papers/2003/dnsspectroscopy/*
[6] *http://public.as112.net/*

ity of the update request. However, the update request may be passing over an insecure channel (the Internet). A one-way hashing function is used to prevent malicious observers from learning the secret key and using it to make their own modifications.

A time stamp is included in the TSIG protocol to prevent recorded responses from being reused, which would allow an attacker to breach the security of TSIG. This places a requirement on dynamic DNS servers and TSIG clients to contain an accurate clock. Since DNS servers are connected to a network, the Network Time Protocol may be used to provide an accurate time source.

Although TSIG is widely deployed, there are several problems with the protocol:

- It requires distributing secret keys to each host which must make updates.
- The HMAC-MD5 digest is only 128 bits.
- There are no levels of authority. Any host with the secret key may update any record.
- As a result, a number of alternatives and extensions have been proposed.
- RFC 2137 specifies an update method using a public key "SIG" DNS record. A client holding the corresponding private key can sign the update request. This method matches the DNSSEC method for secure queries. However, this method is deprecated by RFC 3007.
- In 2003, RFC 3645 proposed extending TSIG to allow the Generic Security Service (GSS) method of secure key exchange, eliminating the need for manually distributing keys to all TSIG clients. The method for distributing public keys as a DNS resource record (RR) is specified in RFC 2930, with GSS as one mode of this method. A modified GSS-TSIG – using the Windows Kerberos Server – was implemented by Microsoft Windows Active Directory servers and clients called Secure Dynamic Update. In combination with poorly configured DNS (with no Reverse Lookup Zone) using RFC 1918 addressing, reverse DNS updates using this authentication scheme are forwarded en masse to the root DNS servers and increase the traffic to root DNS servers in the course of doing so [5]. There is an anycast group which deals with this traffic to take it away from the root DNS servers [6].
- RFC 2845, which defines TSIG, specifies only one allowed hashing function HMAC-MD5, which is no longer considered to be highly secure. In 2006, proposals were circulated to allow RFC 3174 *Secure Hash Algorithm (SHA1)* hashing to replace MD5. The 160-bit digest generated by SHA1 should be more secure than the 128-bit digest generated by MD5.
- RFC 2930, which defines TKEY, a DNS Record used to automatically distribute keys from a DNS server to DNS clients.
- RFC 3645, which defines GSS-TSIG, uses gss-api and TKEY to automatically distribute keys in gss-api mode.
- The DNSCurve proposal has many similarities to TSIG.

**PAUL AMMANN**

***Paul Ammann is a Communication and Security Engineer for a utility company. He is currently working on a DNSSEC book for No Starch Press. He's a pretty ordinary guy and can be reached via email at pq (underscore) aq (at) fastmail (dot) com.***

solutions

# Open Source Systems
# Design - Administration - Consulting

- design and administration of server farms
- load balancing and high availability solutions
- ZFS file servers and storage appliances
- FreeBSD, OpenIndiana and Linux

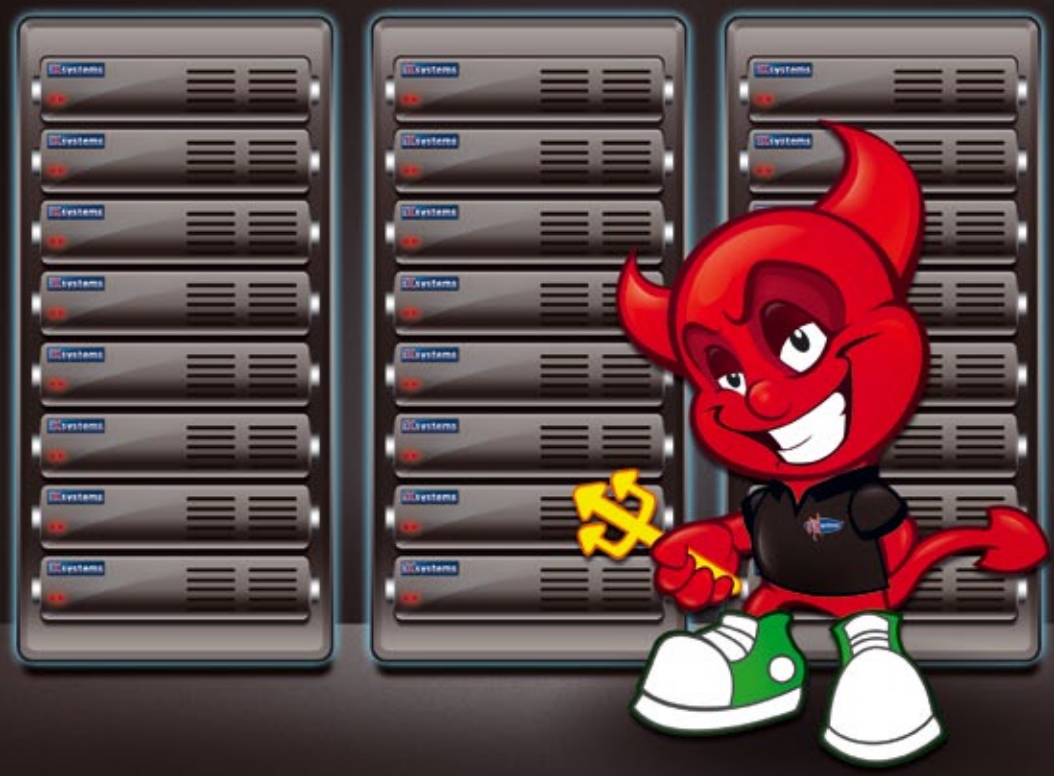
Our open source projects:

VX ConnectBot: SSH and telnet client for Android
mfsBSD: memory-resident FreeBSD installations
zfs-stats: ZFS statistics tools

Contact information:

VX Solutions s. r. o.
Mag. Martin Matuška
E-Mail: office@vx.sk
Web: http://www.vx.sk

# What has your server vendor done for BSD lately? Probably, not much.



## Work with a vendor that supports the operating system you love!

iX is the corporate sponsor of the PC-BSD® Project, a major corporate donor to the FreeBSD Foundation, and leads the FreeNAS™ development team -- all while employing some of the most brilliant minds in the FreeBSD® community. For BSD hardware and software expertise, look no further.

**1-855-GREP-4-IX**

**http://www.iXsystems.com/community**

iX systems®